THE REAL TIME IMPLEMENTATION OF A SPEECH CODING ALGORITHM

Anthony Mark Davis

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg, 1986

DECLARATION

I declare that this project report is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

(Signature of Candidate)

5 day of March 1986

ABSTRACT

This project report describes the real time implementation of a speech coding algorithm on a state-of-the-art digital signal processor.

The coding algorithm utilized is a medium to high complexity 'Hybrid' algorithm originally proposed by the National Electrical Engineering Research Institute (NEERI).

In this report the algorithm is functionally decomposed into five distinct portions, namely, silence extraction, segmentation, Fourier transform, segment comparison and extrema coding. The implementation of each of these portions on the Texas Instruments TMS 32020 digital signal processor, as well as the modifications required for real time performance, are described.

The algorithm was found to execute, at worst case, in 53% of available real time. Preliminary tests of the resultant speech quality indicated favourable results.

ACKNOWLEDGEMENTS

I would like to thank Pierre La Garde of NEERI for entrusting me with the task of implementing his algorithm and for his co-operation and help.

To Professor Hanrahan for his advice and guidance and to Dr. Johann Leitner and the staff of Spescom (Pty) Ltd. for providing the funds and facilities for the project.

CONTENTS PAGE

DECLARATION		ii
ABSTRACT		iii
ACKNOWLEDGEMENTS		iv
CONTENTS		v
LIST OF FIGURES		viii
LIST OF TABLES		ix
1	INTRODUCTION	1
1.1	The Aims of the Project	2
1.2	Scope of the Project	3
1.3	Structure of this Report	3
2	SPEECH CODING TECHNIQUES AND IMPLEMENTATIONS	5
2.1	Introduction	5
2.2	Possibilities of Reducing the Bit Rate	5
2.3	Speech Perception	5
2.4	Human Speech Production	7
2.5	The Six Principles of Speech Encoding	8
2.6	A Review of Speech Compression Algorithms	8
2.6.1	Pulse Code Modulation	9
2.6.2	Differential Quantization	10
2.6.3	Vocoders	12
2.6.4	Sub-band Coders	15
2.6.5	Adaptive Transform Coders	16
2.6.6	Vector Quantization	16
2.7	Summary	17
3	AN OVERVIEW OF THE NEERI SPEECH CODING ALGORITHM	18
3.1	Introduction	18
3.2	Silence Extraction	19
3.3	Segmentation	20
3.4	The Fourier Transform	20

CONTENTS		PAGE
3.5	Segment Comparison	21
3.6	Extrema Coding of the Amplitude Spectrum	22
3.7	Short Coding of the Amplitude Parameters	24
3.8	Decoding	25
3.9	Results of the NEERI Simulations	25
3.10	Summary	26
4	THE DIGITAL SIGNAL PROCESSOR	27
4.1	Introduction	27
4.2	The Choice of Digital Signal Processor	28
4.3	A Review of Available DSP's	28
4.4	Reasons for the Choice of the TMS 32020	30
4.5	The TMS 32020 DSP	30
4.6	Summary	32
5	CODING ALGORITHM IMPLEMENTATION	33
5.1	Introduction	33
5.2	The Software Design Environment	33
5.3	The Software Development Methodology	35
5.4	Program Description	36
5.5	Silence Extraction	37
5.5.1	Program Flow and Variables Used	39
5.5.2	Evaluation of the Program	43
5.6	Segmentation	44
5.6.1	Voiced/Unvoiced Algorithm	45
5.6.2	Pitch Period Determination	46
5.6.3	Program Flow and Variable Allocation	51
5.6.4	Analysis of the Segmentation Routine	72
5.7	The Fourier Transform	73
5.7.1	Implementation of the Transform Portion	77
5.7.2	Segment Length Adjustment	79
5.7.3	The PFA Algorithm Implementation	83
5.7.4	The Square Root Algorithm	83
5.7.5	Analysis of Results	85

CONTENTS

		PAGE
5.8	Segment Comparison	86
5.8.1	The Implementation	87
5.8.2	Analysis of the Results	90
5.9	Extrema Coding	90
5.9.1	The NEERI Approach	90
5.9.2	The Wall ALgorithm	91
5.9.3	Implementation of the Wall Algorithm	95
5.9.4	Analysis of Performance	97
5.10	Summary	97
6	CONCLUSION	99
6.1	Overview of Progress	99
6.2	Analysis of Execution Time and Speech Quality	99
6.3	Achievements of this Project	100
6.4	Future Work	101
REFERENCES		102

LIST OF FIGURES

Figure		Page
2.1	Simplified model of speech production	7
2.2	Autocorrelation functions for speech signals	11
2.3	Block diagram of a delta modulation system	12
2.4	Block diagram of channel vocoder analyzer	13
2.5	Block diagram of sub-band coder	15
3.1	Voiced speech waveform	21
3.2	Extrema Coding	23
3.3	Partitioning of dynamic range for Short Coding	24
5.1	Silence Coding	44
5.2	Distribution of zero crossings	45
5.3	Voiced and Unvoiced areas as a function of the amplitude level and the number of zero crossings	46
5.4	Quasi-periodic nature of pitch	47
5.5	Coding Algorithm task structure	48
5.6	Segmentation algorithm	49
5.7	Segmentation at transition from voiced speech	50
5.8	Calculation of peak estimate	67
5.9	Results of segmentation algorithm	72
5.10	Comparison of actual spectrum with the spectrum attained by resampling and zero padding	80
5.11	Extrema coding using the Wall algorithm	94

LIST OF TABLES

Table		Page
5.1	The time in milliseconds to calculate a length N DFT	75
5.2	The range of values of the PFA using the WFTA prime factor modules 2,3,4,5,7,8,9 and 16	76

CHAPTER 1 : INTRODUCTION

There are many advantages that can be gained by representing speech signals in digital form. In transmission the use of digital regenerative repeaters can make the degradation in long distance transmission negligible. Digital signals are easy to handle in modern electronic switching systems. The use of digits for speech enables telegraphy, speech and data to be intergrated into one transmission network. In military communication the digital representation greatly eases the problem of providing communication security. In speech storage and retrieval applications the ability to use large random-access digital stores in computers gives great flexibility.

The aim of digital coding techniques is to provide speech of acceptable quality, in a digital form, at the lowest bit rate possible.

Digital coding of speech waveforms is by no means a new technology. Numerous signal processing techniques for taking advantage of properties of speech production and perception have been proposed for the purpose of reducing the required transmission rate or storage for speech waveforms.

These techniques range from "low complexity" designs to "high complexity" and they offer a corresponding tradeoff between performance and cost.

The "low complexity" designs can be defined as those in which one or more coders or decoders can potentially be realized in a single chip. Examples of such systems are ADM (adaptive delta modulation) and ADPCM (adaptive differential pulse code modulation) and they typically require bit rates in excess of 24-32 kbits/s for good performance. On the opposite side of the scale are the "high complexity" algorithms such as ATC (adaptive transform coders) and APC (adaptive predictive coders) which have the potential of achieving good quality at bit rates down to 9,6

kbits/s. In the middle are the "medium complexity" designs such as SBC (sub-band coding) and LPC (linear predictive coding) techniques which can provide acceptable speech reproduction at bit rates as low as 1200 bits/s. These techniques are limited by their simple speech production models.

Until recently most studies of coding algorithms, particularly high complexity algorithms, have been restricted to nonreal-time computer simulations due to the difficulty of developing specialized hardware. This stumbling block has now been removed with the introduction of dedicated digital signal processing hardware. These DSP (Digital Signal Processor) chips can now be utilized to implement those techniques which are classed as "medium to high complexity" in real time.

1.1 The Aims of the Project

The aim of this project is to implement a "medium to high complexity" coding algorithm in real time. The algorithm should ideally exhibit good compression characteristics while still preserving intelligibility and speaker recognition. The resulting bit rate for this algorithm should be as low as 4 kbits/s.

The algorithm to be implemented has been proposed by the National Institute of Electrical Engineering Research (NEERI) at the CSIR (Council for Scientific and Industrial Research) and will be refered to as the NEERI algorithm in this report.

The result of this project report will be a set of algorithms implemented on a digital signal processor that will demonstrate that the NEERI algorithm can be implemented in real time. In addition investigations will be conducted into the quality of reproduced speech and the compression performance of the algorithm.

1.2 Scope of the Project

This project report details the first phase of the design of a digital storage and regeneration system. This phase is intended to demonstrate that the required compression algorithm can be implemented in real time. The algorithm will thus be executed and tested in a simulation environment. The simulation will provide information as to the execution time and data generated by each implemented portion of the algorithm.

The software design has been simplified by the functional decomposition of the algorithm into a number of distinct modules. Each of these modules will be coded and tested individually. Due consideration will be given to the intermodule interface so as to allow for the eventual integration of the modules into a single executable program. This approach will aid the debugging, testing and maintenance of the system software.

The development of a compression algorithm is beyond the scope of this project. However, changes or modifications to the NEERI algorithm, necessary to achieve the real time implementation objectives, will be required.

1.3 Structure of this Report

This project report presents the real time implementation and modification of the NEERI algorithm for speech bit rate compression. Chapter 2 provides a detailed description of the possibilities of reducing the bit rate and an introduction into the properties of speech production and perception. A knowledge of these phenomena is required to appreciate the philosophy of the NEERI coding algorithm. A discussion of the state of the art in coding algorithms with emphasis on real time implementations of the techniques will then be presented.

Chapter 3 considers the strategy behind the NEERI algorithm. This chapter serves to introduce the reader to the overall philosophy behind the algorithm without discussing the detail. It should once again be emphasised that a number of changes have been made to the original proposals by NEERI and thus a detailed coverage of the algorithms employed in the implementation are contained in chapter 5.

Chapter 4 focuses on the state of the art in digital signal processors (DSP's). A number of currently released DSP's are discussed with emphasis on their capabilities, availability, and features. A choice of DSP is then motivated according to the requirements of the project.

Chapter 5 describes the customised software development environment and the software design philosophy. The algorithm is functionally decomposed into modules and the software design and implementation of each of these modules is described. The performance of each module is then analyzed from the point of view of execution time and accuracy.

Chapter 6 concludes the report by considering the results of the simulations. Details of tests of the algorithm on certain speech phrases is presented. Proposals for additions and modifications to the algorithm and implementation are made.

CHAPTER 2 : SPEECH CODING TECHNIQUES AND IMPLEMENTATIONS

2.1 Introduction

This chapter presents a survey of a number of speech coding techniques and examples of their implementation in real time. These techniques are discussed in the light of the theory of speech propagation and perception and with consideration of the 'six principles of speech coding' [O'Neal, 1983].

2.2 Possibilities for Reducing the Bit Rate

For any ordinary audio communications channel it is possible to calculate its communication capacity in terms of the bandwidth of the channel and the degree of accuracy to which the signal can be specified within that bandwidth [Shannon, 1948]. For a typical telephone channel of bandwidth 3 kHz and signal to noise ratio (SNR) of 40 dB, the capacity is 40 000 bits/s. For digitization based on quantizing to a set of equally spaced levels, the number of bits per sample needs to be slightly more than the number implied by the SNR if there is to be negligible degradation of the signal. This would imply a bit rate of approximately 60 000 bits/s.

On the other hand the human cognitive processes cannot process any information that exceeds a few tens of bits per second [Webster, 1961]. This implies a ratio of information transmitted to information used of between 1000 and 10 000. This very large ratio suggests that the full information capacity of an audio channel is not necessary for speech transmission.

2.3 Speech Perception

The amount of information transmitted can be reduced by careful consideration of the physiological constraints of the human auditory system.

The inner ear can be regarded as an approximation to a very large number of fairly broad bandpass filters with extensively overlapping pass bands. On the output of each filter is a highly non-linear 'hair cell' connected to a nerve fibre carrying the response, coded in the form of frequency and timing of nerve pulses to the higher auditory centres for further processing.

Pitch perception of periodic signals is much more precise than would be expected from the filtering action of the inner ear and is probably connected with the fact that nerve pulse intervals tend to synchronise to multiples of the repetition period of the sound.

The ear is not appreciably affected by phase distortion if the pressure wave pattern applied to the individual hair cells is not much disturbed. This implies that group delay distortion is not noticed if it varies little within one critical bandwidth. However, human speech is normally heard in the presence of significant time dispersion caused by reverberation in the acoustic environment and so a much larger random group delay distortion is not detected as unnatural if it is within the range that occurs naturally.

A further consequence of the ears filtering action is the ability of high intensity sounds in one frequency region to mask lower level sounds in other regions. This property is of great value in making the listener insensitive to low-level background noise or distortion when the wanted signal is of a higher level.

To summarise, most auditory experiments conclude that acceptable voice communication is achieved by the preservation of the short-time amplitude spectrum of the speech signal. Some loss of phase information is permissible and this implies that exact reproduction of the speech waveform is not necessary for acceptable voice communication .

2.4 Human Speech Production

Additional reduction in bit rate can be achieved when designing a coding system by making use of the fact that the signal is produced by a human talker. The physiology and habits of use of the speaking mechanism put strong constraints on the types of signal that can occur.

Speech is generated by exciting an acoustic cavity, the vocal tract, by pulses of air released through the vocal cords for voiced sounds or by turbulence for unvoiced sounds. A single and useful model for speech production consists of a linear system representing the vocal tract, driven by an excitation function which is a periodic pulse train for voiced sounds and wide-band noise for unvoiced sounds (see Fig 2.1). The response of the linear system is of a resonant nature, so that its transfer function is characterised by a set of resonant frequencies, referred to as formants. If the excitation ... al tract parameters are fixed the speech spectrum ... velope representing the vocal tract transfer funct... a fine structure representing the excitation.

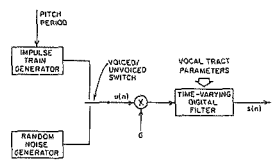

Figure 2.1 Simplified model of speech production.

Speech has a well defined short-time spectrum and as a result the linear system can be represented as a slowly time-varying system and on a short-time basis it is approximately stationary.

2.5 The Six Principles of Speech Encoding

From the above description of speech production and perception it appears that bit rate compression can effectively be achieved by only transmitting the information that the ear can perceive. Alternatively, the relevant information contained in the model of speech production could be extracted and the speech could be reproduced by means of the model. O'Neal [1983] has postulated six principles of low bit rate speech encoding which extract most of the redundancy inherent in the speech waveform. The six principles are :

1. Remove the temporal redundancy in the signal
2. If the signal is not present in a time interval, do not transmit it
3. If the signal is not present in a frequency interval do not transmit it
4. Assign long code words to infrequently transmitted messages and short codewords to frequently transmitted messages
5. Shape the quantizing noise spectrum so that it is least objectionable to the listener
6. Adapt the coding algorithm to the changing statistics of the signal

The effectiveness of coding algorithms can be determined by the considering them with respect to the above six principles.

2.6 A Review of Speech Compression Algorithms

In this section the theory and details of existing implementations, of a number of speech coding algorithms is presented. It is not the intention of the author to motivate the choice of algorithm for this project as the implementation of the NEERI algorithm is the primary objective of this project. The details of other existing techniques is presented to put the reader in perspective as to the state of the art algorithms and

methods of implementation.

Speech coding algorithms can be grouped into three main catagories, namely, waveform coding, hybrid coding and analysis-synthesis systems.

Waveform coders, as their name implies, attempt to copy the actual shape of the speech waveform. Waveform coding systems are used to code speech waveforms at bit rates of 12 kbits/s to 64 kbits/s.

Analysis-synthesis systems involve the use of some model of the speech generating mechanism. The reduction in bit rate is a result of assuming that the parameters of the model are changing slowly and the receiver needs to be kept abreast of only the changes. Analysis synthesis systems such as channel vocoders, formant vocoders and LPC (linear predictive coders) can achieve bit rates of less than 5 kbits/s.

Hybrid coders are intermediate systems that combine techniques of waveform coding and analysis-synthesis coding. These systems normally give better speech reproduction in the 4 to 16 kbit/s range than is possible with either of the other two classes of system, at these bit rates.

2.6.1 Pulse Code Modulation

The fundamental theory of the digital representation of speech waveforms states that it is theoretically possible to reconstruct an analog waveform exactly if the waveform was sampled at a rate that exceeds twice its bandwidth. The samples of the signal waveform can then be represented in a digital code with enough digits to specify the signal co-ordinates sufficiently accurately. This principle of coding, known as Pulse Code Modulation (PCM) was suggested by Reeves [1938] and is currently widely used. This simple technique does not exploit any of the special properties of speech production or auditory perception,

except their limited bandwidth. The bit rate of 4 kHz bandwidth speech using 8 bits/sample is 64 kbits/s.

The amplitude of the quantizing noise for simple PCM is determined by the step size. During low level speech or silence this noise might be very noticeable, whereas, during loud speech it would to some extent be masked by the wanted signal. For a given degradation it is therefore permissible to allow the quantizing noise to vary to some extent with signal level. This variation can be achieved either by using a non-uniform distribution of quantizing levels or by making the quantizing step size change as the speech level varies (Adaptive PCM). Both these methods have been adopted and useful communications performance can be achieved at 4 bits/sample.

Non-uniform quantizers such as logarithmic quantizers have been effectively implemented in hardware and are in everyday use in the telecommunication field. The above techniques are unable to yield further reduction in bit rate as they simply exploit the knowledge of the amplitude distribution of speech. In the next section techniques will be described that utilize the very high sample to sample correlation in speech waveforms to further reduce the bit rate.

2.6.2 Differential Quantization

Figure 2.2 shows that there is a very high correlation between the adjacent speech samples as well as significant correlation between samples that are several sampling intervals apart. This implies that the signal does not change significantly from sample to sample so that the intersample difference should have a lower variance than the samples themselves. Basically, differential quantization can be described as determining the differences, quantizing them, and recovering an approximation to the original signal by intergrating the quantized differences.

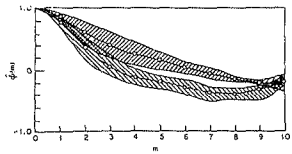

Figure 2.2 Autocorrelation functions for speech signals; upper curves for lowpass speech, lower curves for bandpass speech

The simplest form of differential quantizer is the Delta-modulator. A block diagram is shown in Fig. 2.3. The Delta-modulator uses only one bit per sample and merely indicates whether the resulting waveform is to be increased or decreased by one step. This type of coder offers the possibility of extremely simple hardware implementation and single chip solutions are available. The performance of these delta-modulators can be further enhanced by making the step size adaptive. There are a variety of Adaptive Delta Modulation (ADM) techniques and bit rates as low as 16 kbits/sec, for communication quality speech, are possible.

The advantages of delta-modulation can be combined with those of PCM if a PCM coder is used instead of a one bit quantizer in the feedback loop. Further improvements can be realized by including techniques for level adaption. This method, known as Adaptive Differential PCM or ADPCM gives excellent quality speech of telephone bandwidth at 32 kbits/s. At lower rates such as 16 kbits/s, the quantizing noise is noticeable but slightly less objectionable than the noise given by ADM or APCM at the same digit rate [Jayant, 1974].

The ADPCM algorithm has been successfully implemented using using custom Digital Signal Processors [Crochiere et al, 1982] and recently on the TMS 32010 digital signal processors.

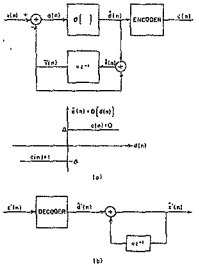

Figure 2.3 Block diagram of a delta modulation system; (a) coder; (b) decoder

2.6.3 Vocoders

Vocoders depend upon a rigid parameterization of the speech signal in accordance with a linear quasi-stationary model of speech production. The traditional model, shown in Fig. 2.1 is one where the source and the resonant system that modulates the sound are separable and do not interact.

The speech signal is analysed over a time window of a few tens of milliseconds. This signal has either a periodic (for voiced sounds) or a random (for unvoiced) time structure. In the frequency domain the fine structure of the spectrum consists either of lines resulting from the harmonics of the fundamental frequency (the pitch) or is continuous during random noise

excitation. The envelope of the speech spectrum is the result of combining the spectral trend of the sound source with the response of the vocal tract.
By separating the fine structure specification of the sound sources from the overall spectral envelope and specifying both in terms of a fairly small number of slowly varying parameters, it is possible to transmit a reasonably adequate description of the speech at data rates of 1000 to 3000 bits/s.

At the recieving or reproduction end the speech is synthesised by using either a periodic or random noise source to drive a dynamically controlled spectral shaping filter.

There are many types of vocoders each differing from the other generally in the means of specifying the variable spectral shaping filter. The three main catagories are the channel vocoders, linear predictive coders and formant vocoders.

Channel Vocoder

In the channel vocoder the spectrum is represented by the response of a bank of contiguous variable-gain bandpass filters. The desired response can then be approximated using separate contributions from the individual channels as illustrated in Fig. 2.4.

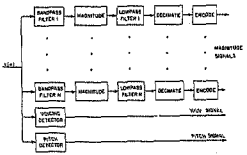

Figure 2.4 Block diagram of channel vocoder analyzer

Linear Predictive Coder Vocoder (LPC vocoder)

The LPC coder provides the spectral approximation by means of the response of a sampled-data filter whose all-pole transfer function is chosen according to a least-squared-error criterion. LPC techniques provide bit rates in the region of 2400 bits/s with reasonable quality.

Numerous implementations of LPC techniques have been proposed (see Maitra and Davis [1979], Gold and Tierney [1983], Miyamoto et al [1983]).

LPC analysis is successful provided that the overall speech spectrum approximates the response of an all-pole filter. This is, however, not always the case especially with nasalised vowels and consonants. Under these conditions the LPC synthesis produces spectral peaks with large bandwidths and an associated 'buzziness' in speech quality [Holmes, 1982].

Formant Vocoders

Formant vocoders utilize a spectral filter system that has resonators that are explicitly related to the principal formants of the input speech. The coding system is thus constrained to deal only with the known frequency range and necessary accuracy specification of each formant. The true bandwidths of the formants do not vary much during speech and such variation as does occur is fairly predictable. The preservation of the actual formant bandwidths is not perceptually important and this property of the resonances is not usually transmitted in formant vocoders. Data rates of 1200 bits/s have been achieved using these techniques.

Quarmby and Holmes [1984] have implemented a parallel formant synthesiser on the NEC digital signal processor.

2.6.4 Sub-band Coders

In the Sub-band coder the speech band is typically divided into four or more sub-bands by a bank of bandpass filters. Each of these sub-bands is frequency translated to zero frequency, it is then sampled (or resampled) at its Nyquist rate (twice the bandwidth) and digitally encoded with a PCM of DPCM coder (see Fig. 2.5). In this process each sub-band can be coded according to perceptual criteria that are specific to that band.

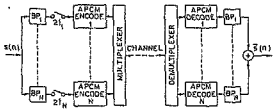

Figure 2.5 Block diagram of sub-band coder

On reconstruction, the sub-band signals are decoded and modulated back to their original frequencies. They are then summed to give a replica of the original speech signal.

The speech quality produced by the sub-band coders in the 16 to 32 kbit/s range is much improved over that possible with the best conventional waveform coders at the same bit rates.

The most complex part of this coder is the filter bank. Destinct advantage can be gained by implementing this coder digitally and thereby taking advantage of quadrature-mirror filters. Crochiere et al [1982] describe the implementation of two,four and five band SBC coders utilizing the 'Bell Labs' digital signal processor.

2.6.5 Adaptive Transform Coders

This 'high complexity' technique involves the block transformation of windowed input segments of the speech waveform. Each segment is represented by a set of transform coefficients which are separately quantized. On reproduction the quantized coefficients are inverse transformed to produce a replica of the original input segment.

The most common type of transform for speech processing applications is the time-to-frequency transformation. An example of this is the Discrete Cosine Transform (DCT) which has been found to be well suited to speech processing.

Allocation of bits to code each transform coefficient can be decided adaptively. By careful choice of the bit allocation it is possible to get a subjectively very close approximation to telephone speech with 16 kbits/s and useful communication quality with only 9,6 kbits/s.

2.6.6 Vector Quantization

The Vector Quantization coding process is a pattern matching technique. Each vector is encoded by comparison with a set of stored reference vectors, known as codevectors or patterns. Each pattern will be used to represent input vectors that are somehow identified as 'similar' to this pattern. The best matching pattern in the codebook, the set of stored reference patterns, is selected by the coding process according to a suitable fidelity factor and a binary word is used to identify this pattern in the 'codebook'.

The signal vector can be either a segment of samples, in the case of a PCM signal, or parameters extracted from a signal for LPC applications. For the example of LPC coded speech, bit rates as low as 800 bits/s have been achieved in laboratory simulations.

The real time implementation of Vector Quantization systems has been a problem due to the amount of computation required to implement pattern recognition algorithms as well as the limitations on the amount of memory that can be utilized to store the 'codebook'.

2.7 Summary

In this chapter the theory of speech production and perception has been presented. This has emphasized the fact that bit rate compression can be achieved by the knowledge that a human is producing the speech and ultimately a human will hear the speech. The six principles of speech encoding were stated and a number of classical techniques for bit rate reduction were discussed.

CHAPTER 3 : AN OVERVIEW OF THE NEERI SPEECH CODING ALGORITHM

3.1 Introduction

In this chapter the overall philosophy behind the NEERI coding algorithm will be presented. The purpose of this chapter is not to describe the details of the coding algorithm rather it attempts to familiarize the reader with the overall coding strategy employed. It should, however, be stressed that the final version of the algorithm as presented in this report is a modified version of the NEERI algorithm.

This algorithm has been proposed as a solution to the problem of speech storage and as a result little consideration is given to the effects of channel errors on the resultant speech. It is the opinion of the author that this algorithm in the form detailed below is not suitable for transmission applications due to its susceptibility to bit errors.

The NEERI algorithm is essentially a 'Hybrid type' algorithm which attempts to remove the inherent redundancy in the input speech signal. The algorithm can be further catagorized by describing it as a 'high complexity' coding algorithm. The algorithm has been designed with the following objectives in mind:

- the removal of the silence portions of the speech
- the removal of the temporal redundancy of the speech waveform
- the extraction of relevant frequencies from the amplitude spectrum
- the adaption of the coding strategy to the dynamic nature of the data

The algorithm has been functionally decomposed into a number of distinct sequential portions. Each of these portions has been implemented, using simulation techniques, by the personnel at NEERI. The portions will be described below in the light of these simulations. The problems associated with the real time implementation as well as the modifications to this algorithm will be detailed in chapter 5.

3.2 Silence extraction

Speech signals typically contain silence both during a sentence (between successive words) as well as between successive sentences. In accordance with O'Neals six principles of speech encoding, the silence signal itself does not contain any useful information and therefore need not be transmitted. Moreover, silence detection followed by coding contributes significantly to the attainable compression ratio. The only relevant parameter of the silence necessary for accurate reconstruction of the original speech waveform, is its duration.

In the NEERI algorithm the silence is extracted from the input signal and replaced by a silence codeword along with a duration parameter (calculated by counting the number of samples that constitute the silence portion). The silence is then reconstructed by inserting 'zeros' in the reconstructed waveforms. The amount of 'zeros' inserted is determined from the duration parameter.

Silence is detected once a predetermined number of samples fall below a dynamic silence level. The silence level is updated periodically by considering the average level of the conversation (from the conversation history) and a fixed speech to silence ratio. Conversely, the silence mode is complete once a fixed amount of samples exceeds the silence threshold (see section 5.4 for more details).

3.3 Segmentation

The function of this portion of the algorithm is to partition or segment the speech waveform into pitch periods and classify each segment as being either voiced or unvoiced. This is a crucial portion of the NEERI algorithm as good segmentation implies good compression (see Section 3.4). The method used is a time domain peak picking technique. This technique avoids the computationally expensive auto-correlation and inverse filtering algorithms [Rabiner et al,1976] used in the various types of vocoders.

The primary reason for performing pitch period segmentation is to yield a good approximation of the short time speech amplitude spectrum (see Section 3.2). Pitch synchronous analysis eliminates the spectral effects of the window and driving functions and results in a 'clean' amplitude spectrum. Further advantages of pitch synchronous analysis include the use of a rectangular window when transforming to the frequency domain (i.e. no computation of window weighting factors). In addition the approach avoids the problem of pitch in the frequency domain by coping with it in the time domain.

3.4 The Fourier Transform

Each resulting speech segment is transformed to the frequency domain via the Fourier Transform. The real and imaginary parts of the complex spectrum are combined to yield the magnitude (or amplitude) spectrum and the phase information. As stated in Section 2.2, the ear is sensitive to the parameters of the amplitude spectrum and not to the relative phase of the frequency components. The phase information is thus redundant and can be disposed of at this stage. The result is the elimination of half the segment data points.

3.5 Segment Comparison

The aim of this portion is to remove the temporal redundancy inherent in the speech waveform (usually in the voiced parts). A visual analysis of any voiced portion of a speech waveform clearly shows the similarity between successive pitch periods. This is due to the fact that the elements of human speech production (vocal tract, tongue position, etc) change slowly. This results in a smooth transition in both pitch and vocal tract transfer function. This accounts for the relative similarity of succeeding pitch periods in both response and duration (see Fig. 3.1).

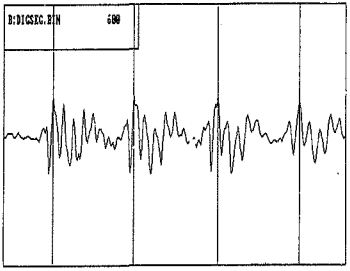

Figure 3.1 Voiced speech waveform

The amplitude spectrum of each pitch period is compared with the spectrum of the preceding period. If the deviation between the two segments is less than a certain value the segments are assumed to be identical and a 'segment repeat code' is stored in place of the amplitude spectrum. During the speech reproduction process the 'segment repeat code' is replaced by the amplitude spectrum of the preceding segment.

The total number of successive segment repetitions is limited to three, as an 'echo' effect is introduced if abundant repetition takes place.

3.6 Extrema Coding of the Amplitude Spectrum

The speech spectrum consists of a number of peaks called formants. The ear is sensitive to these formants and can tolerate the following variation of formant parameters [van Schalkwyk,1985] :

- a 3% variation in the formant frequency
- a 30% variation in the formant bandwidth
- a variation of 1,5 - 3,0 dB in formant amplitude

This implies that a suitable approximation (within the above limits) of the amplitude spectrum would suffice.

The extrema coding algorithm attempts to approximate the magnitude spectrum of each segment by means of a series of straight line approximations. Each of the lines that constitute the spectrum approximation are constructed in the following manner :

- a line is drawn from the start sample (of the spectrum) to the next sample a distance of 2 sample periods away (see Fig. 3.2(a))
- the distortion between the original portion of the spectrum and the straight line approximation is then calculated
- another straight line is then constructed from the start sample to the sample 3 sample periods away (see Fig 3.2(b))
- the distortion of this approximation is then calculated
- this process is repeated until 18 sample periods separate the start sample from the endpoint sample of the straight line
- the closest straight line approximation is then taken as the approximating line which has a distortion measurement

below a preset value and the greatest distance from the start sample (see Fig 3.2(c) and Fig. 3.2(d)).

- the sample that constitutes the endpoint of the straight line approximation is then coded as an extrema and thereafter becomes the start sample for the next straight line approximation.

- each extrema is represented by two parameters, namely, its magnitude and a parameter called the Time Between Extrema (TBE) which contains the temporal distance between the start sample and the extrema (or endpoint sample)

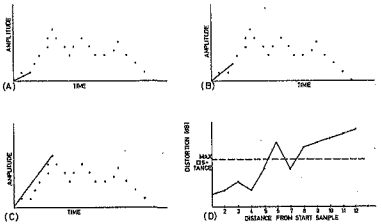

Figure 3.2 Extrema coding (a) first straight line approximation (b) second approximation (c) best approxiamtion and (d) distortion vs. distance curve for the segment. Note the choice of the sample 7 periods from the start sample

The method of distortion calculation used includes information concerning the amplitude of the portion of the spectrum under consideration. This is done so as to code high level samples (usually the important formants) accurately and approximate the low level frequencies roughly.

The amplitude and separation information is used to decode the extrema's by linear interpolation to yield an approximation of the amplitude spectrum [LaGarde,1985].

3.7 Short Coding of the Amplitude Parameters

This portion of the algorithm codes the amplitude parameters of the extrema points with less bits than originally used. An adaptive logarithmic coding technique is utilized that divides the dynamic range of the input data into a number of blocks (see Fig 3.3). Each segment is analysed to determine the maximum amplitude value. This value is then used to select an appropriate coding table for the segment. All of the amplitude samples are then quantized according to this table.

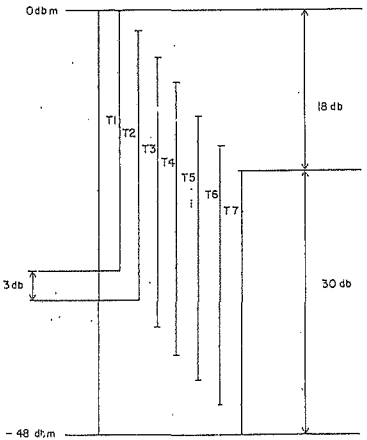

Figure 3.3 Partitioning of dynamic range for Short coding

Each of the tables has the same dynamic range and therefore each data point is represented by the same number of bits (regardless of the table used).

During the coding process the table number is appended to the segment control code to enable decoding of the short coded amplitude samples.

3.8 Decoding

The decoding algorithm reverses the coding process and synthesizes the original speech, however, due to the loss of phase information during the Fourier Transform portion, there will be no correlation between the original and the synthesised speech waveforms (although they should sound the same).

The decoding algorithm simply performs the coding in reverse, but it is far simpler than the coding algorithm as Silence extraction can be replaced by a simple zero volt insertion algorithm, the segmentation procedure is obviously not included and the extrema coding algorithm is replaced by a straight line interpolation algorithm.

The only unique portion of the decoding schema will be the phase adjustment which is necessary to account for end effects during segment recombination in the time domain.

3.9 Results of the NEERI Simulations

The above NEERI algorithm has been simulated on a VAX 730 minicomputer using FORTRAN coded programs. At the commencement of this implementation, study preliminary tests of the audible quality of a time domain version of the algorithm had been conducted. These tests provided clear understandable speech at 8 Kbits/s. This version of the algorithm did not, however, perform the Fourier Transform and as a result the extrema coding was

performed on the time domain signal rather than its magnitude spectrum.

Analysis of the potential compression characteristics of the frequency domain version of the algorithm described in this chapter indicate, that bit rates as low as 2 Kbits/s could be attained without much loss in speech quality. This fact together with the potential noise robustness of the algorithm provided suitable grounds for initiating this study.

3.10 Summary

In this chapter the overall coding philosophy behind the NEERI coding algorithm has been presented. The algorithm was partitioned into a number of sequential operations which included the silence coding, segmentation, Fourier transform, extrema coding and short coding. The aim of each of this portions of the algorithm was discussed.

CHAPTER 4 : THE DIGITAL SIGNAL PROCESSOR

4.1 Introduction

The implementation of many of the algorithms and techniques described in chapter 2 has been severely limited by the lack of appropriate high performance processors capable of implementing the complex mathematical operations required by typical signal processing applications. Until recently these techniques have been implemented using complex bit slice processors, dedicated hardware and conventional micro-processors which are not suited to the demands of digital signal processing.

The Digital Signal Processor (or DSP) has been designed to cater specifically for the requirements of digital signal processing algorithms. Thompson et al [1982] stated that '... the essential elements of a digital signal processor can be deduced from a careful consideration of the algorithms it must execute.' The result is a host of devices each providing various features essential to the efficient execution of signal processing algorithms.

The DSP will certainly fill the gap that exists between conventional analog circuitry, which suffers from inherent performance limitations and other drawbacks, and the expensive-to-implement building block DSP components: multipliers, high speed memories and support components.

The purpose of this chapter is to provide the reader with an overview of the state-of-the-art in digital signal processors so as to appreciate the features and complexity of the processor chosen to implement the NEERI algorithm.

4.2 The Choice of Digital Signal Processor

The choice of DSP for this project was based on the following criteria:

- eventual cost and local support
- availability of development tools
- performance with respect to the diverse requirements of the NEERI algorithm

The first two criteria effectively eliminate the bulk of the available processors. The third criteria is important as most of the DSP's available have been designed with an algorithm or application in mind.

The NEERI algorithm contains many diverse requirements which extend from the execution of complex Fourier transforms to the manipulation of simple lookup tables. As a result a general purpose DSP is required which provides for both the 'run of the mill' processing as well as specific signal processing techniques.

4.3 A Review of Available DSP's

With the advent of the single chip DSP, system designers are now able to implement many complex signal processing algorithms. This has resulted in a fast pace of change in this area which is best characterised by the rapid evolution of single chip programmable DSP designs.

The DSP era was initiated by the simple architectures of the Intel 2920/2921 and the Gould-AMI's 28211/28212. These devices are now obsolete and are generally in the process of being phased out.

The trend today is to move from traditional von Neumann architectures to modified designs incorporating pipelining. Illustrating this trend is Texas Instruments second generation TMS32020. It is distinguished from the first generation 32010 by its large program and data memory spaces, reconfigurable memory maps, large on chip random access memory, a multiprocessor interface, and an enhanced DSP oriented instruction set. The new design tries to take advantage of standard von Neumann architectures' best characteristics but combines it with some aspects of a highly parallel architecture.

Parallelism is achieved through a combination of techniques including a reconfigurable twin-RAM organization in which a total of 544 by 16 bits of RAM has been split into two separate blocks. The first is permanently mapped into the data memory (288 by 16 bits). The second, which is under program control, can be mapped into cache or program memory (256 by 16 bits).

The TMS32020 achieves a cycle time of 200 ns.

NEC has taken the concept of parallelism in DSP operations a stage further with the use of a non-von-Neumann data flow architecture in its 7281 single chip DSP. This n-MOS chip contains four blocks that supervise control and processing operations and five others that store data addresses and other information. Each control and processing block contains its own set of instructions, which can be executed independently or in conjunction with other instructions from other blocks. With such an architecture it is theoretically possible for the 7281 to operate at up to 5 million instructions per second

In addition to the above DSP processors, a host of varied solutions have been proposed. These range from the special purpose DSP's like the NEC 7764 (which has been specifically optimized for speech recognition) to multi-chip DSP families which allow the system designer the freedom of picking the appropriate building blocks -multipliers, accumulators, barrel

shifters, floating point processors and arithmetic logic units- and connecting them.

4.4 Reasons for the Choice of the TMS 32020

Section 4.1 detailed the requirements of the digital signal processor to be used in the implementation of the NEERI algorithm. The TMS 32020 was chosen for this assignment for the following reasons :

- the TMS 32010 had an impressive track record and was adopted for use in a number of projects
- the local support of Texas Instrument products was considered to be adequate
- a full range of development tools was available. These included cross-assemblers, software simulators and real time in-circuit emulators
- the complexity of the NEERI algorithm warranted a processor capable of addressing a large program RAM
- the algorithm could be implemented on a single TMS 32020
- the projected cost of the chip was expected to be acceptable by the middle of 1986

4.5 The TMS 32020 DSP

In this section the features of the TMS 32020 will be summarized. For detailed information the reader should consult the 'TMS32020 User's Guide' [Texas Instruments,1985].

The TMS 32020 has the following features :

- a 200 ns instruction cycle
- 544 words of on chip data RAM, 256 of which may be configured as either data or program memory
- 128K words of data/program space (64K x 16 bit words of data memory, 64K x 16 bit words of program memory)

- sixteen input and sixteen output channels
- directly accessible external data memory space
- 16 bit data words with internal 32 bit operations
- 32 bit ALU and accumulator
- single cycle multiply/accumulate instructions
- 0 to 16 bit scaling shifter
- fractional/integer arithmetic operations
- bit manipulation and logical instructions
- instruction set support for floating point operations
- block moves for data/program management
- repeat instructions to make efficient use of program space while allowing pipelined operation
- five auxiliary registers for indirect addressing with auto-increment/decrement capability and temporary storage
- dedicated arithmetic unit for operating on auxiliary registers
- serial port for multi-processing or interface to codecs, serial analog-to-digital converters etc.
- on chip timer for control operations
- three external maskable user interrupts
- input pin polled by software branch instruction
- output pin for signalling external devices
- on chip clock generator
- single 5-V supply
- 68 pin grid array package

The features of the TMS32020 detailed above allow for the implementation of a number of DSP algorithms, however, the following features apply directly to this project:

- the 200 ns instruction cycle is extremely fast and as will be shown, allows for the real time implementation of the algorithm
- the 544 words of on chip data RAM is necessary to store the sample data associated with a particular speech segment. This data includes the time and frequency domain samples as well as temporary data. The use of on-chip RAM significantly improves the real time performance of the

algorithms
- the 64K of addressable program memory is more than sufficient for this application
- the directly accessible external data memory allows for storage of infrequently used data.

4.6 Summary

This chapter has presented a review of the features of a number of available digital signal processors. The reasons for the choice of the TMS 32020 was motivated and a number of its pertinent features presented.

CHAPTER 5: CODING ALGORITHM IMPLEMENTATION

5.1 Introduction

This chapter presents a detailed discussion of the software development environment, the design philosophy and the implementation of each portion of the coding algorithm. In all the following sections it is assumed that a 10 bit Analog to Digital converter, with a sampling rate of 12.5 KHz, is used.

5.2 The Software Design Environment

One of the primary objectives of this project was the creation of a software development workstation that would aid the designer in the design and testing of the system software (the system software refers to the TMS 32020 assembler code that performs the tasks required to implement each portion of the coding algorithm). This environment or workstation was designed on the assumption that the details of each portion of the coding algorithm would be available to the designer either in the form of an unstructured high level language program or by means of a document detailing the functional requirements of the particular portion.

With the above constraints in mind it was decided to design a development workstation that would allow for the following features:

- efficient simulation of each portion of the algorithm
- structuring of the software to enhance readability and maintenance
- testing of the TMS 32020 source code
- graphical display of the speech files and the resultant encoded files
- audible testing of the coding algorithm

The following TMS 32020 development tools were purchased, an IBM-PC based cross-assembler and a TMS 32020 simulator. This enabled the assembly of TMS 32020 source code programs and their subsequent testing using the simulator.

A graphics 'toolbox' data analysis package was designed to display and compare binary as well as coded speech files (which contain embedded codewords). In addition a number of interface programs were designed to enable the conversion of binary files to hexadecimal format (for the purposes of the TMS 32020 simulator) and ILS format (to enable use of the 'LSN' command for listening to the speech file).

Each portion of the algorithm was developed as follows:

- the algorithm was coded in Pascal, a structured high level language. This code was designed with the hardware constraints of the TMS 32020 in mind (eg. Arrays were not declared to be larger than the internal data space of the TMS 32020, Real data types were avoided, etc.). The Pascal program was then debugged and tested until it satisfied the required specifications. The result was a high level simulation of the algorithm. The program was then used on a specific speech file and the results of this process were stored as benchmark results for the algorithm
- the TMS 32020 source code was then designed. The code was modelled on the structure and variables used in the Pascal program.
- the source code was then assembled and loaded by the TMS 32020 software simulator. The function of the simulator was to execute the source code and to aid in the debugging of the software. The debugging process continued until the results of the simulation matched those of the Pascal benchmark
- the real time performance of the algorithm was then calculated using the simulator's clock cycle counter

The above process resulted in a program coded on the TMS 32020 which would perform the exact function (with the same accuracy) as the Pascal simulation program. The main problem associated with the TMS 32020 code development cycle was the time taken to assemble, link and simulate each portion of the algorithm. This contrasted sharply with the simplicity of the Pascal development environment and as a result the Pascal programs were used when the algorithm was tested on long speech files.

Comprehensive listings of the TMS 32020 software are contained in the 'SPESCOM Internal Software Documentation Manual', 1985.

5.3 The Software Development Methodology

The coding algorithm was functionally decomposed into a number of portions. This decomposition process was utilized in the development cycle as it enabled each portion of the algorithm to be designed, debugged and tested individually.

Each portion of the algorithm has thus been designed to function as a single entity that reads input data from a specific file and after processing stores the resultant data in an output file. This 'sequential' execution process will not occur in the real time implementation as the individual portions will ultimately be intergrated to form a single executable program.

The envisaged coding program will consist of two main foreground tasks and a number of background tasks. The foreground tasks will consist of the silence extraction and segmentation portions. This will result in the 'packetization' of the input speech, where each packet will consist of one speech segment (a pitch period or multiple thereof, for voiced speech). Each of these packets will then be processed by the background tasks (Fourier transform, segment comparison, extrema coding and short coding) to produce a coded segment.

Each portion of the algorithm has been designed with the eventual integration plan in mind.

5.4 Program Description

All the algorithms and programs detailed in this report will be described by means of a descriptive language program. The descriptive language utilizes the basic structured constructs employed in languages such as Pascal and Ada, however, there will be no formal declarations of variables. Rather, the main variables or symbols used in the implementation will bedescribed prior to the algorithm description.

The descriptive language has been tailored to include the features available on the TMS 32020. The use of auxiliary registers for indirect addressing is incorporated by means of the '@' symbol e.g. to load the data from the location pointed to by auxillary register 1 is represented as follows:

```
.
.
load data @AR1
.
.
```

In addition pipelined incrementing and decrementing of the auxillary register is also supported e.g execute the same instruction as the previous example except increment register AR1 after the operation:

```
.
load data @AR1 and increment
.
```

Many of the other pipelined features of the TMS 32020, such as the barrel shifter, accumulator output shifts and the multiplier facilities are supported by the language and the reader should

have no difficulty in understanding the descriptive expressions that utilize these features.

5.5 Silence Extraction

The function of this portion of the algorithm is to detect the existence of silence in the incoming signal and to extract the silent portion from the signal and code it appropriately.

There are two main criterion upon which a decision between silence and speech can be made in the time domain [Rabiner, 1978]:

- the energy of the speech. Silence has a lower energy than speech utterances
- the short time zero crossing rate. Silence has a lower average zero crossing rate than the unvoiced parts of the signal

Only the first criteria is implemented at this stage, however, information concerning the zero crossing rate is obtained during the segmentation portion and this could ultimately be incorporated once the various portions of the algorithm are integrated.

The energy level of the speech is directly related to the average amplitude. The silence extraction algorithm thus monitors the amplitude of the input samples and depending on a number of criteria a silence/speech decision is made.

The silence extraction algorithm functions as follows; the absolute value of each input sample is compared with a dynamic variable called the 'silence level'. If the magnitude of the sample is less than the 'silence level' a counter known as the 'silence counter' is decremented. If the value exceeds the 'silence level' the counter is incremented.

The transition between speech and silence occurs when the silence counter equals its maximum value (set to 35 by NEERI for 12,5 KHz sampling). The input signal is then classified as silence. The signal is only re-classified as speech when the silence counter is decremented to zero. This method of detection allows for a 35 sample validation time. This prevents the incorrect classification of unvoiced speech as silence, however, it could result in the classification of a number of speech samples as silence (at the transition of silence to speech) or for the same reason silence samples as speech (in the speech to silence transition).

For this reason a 30 sample data buffer is used to store 30 previous input samples. Each new sample is compared with the silence level and stored in the buffer (which is a 'First In Last Out' Type). The silence/speech decision is then made and the 'oldest' sample in the buffer is output if speech is detected. If the input signal is determined to be silence, the 'oldest' sample is removed from the buffer and a sample counter, which keeps track of the number of silence samples detected, is incremented. At the conclusion of the silence detection this sample count is output along with a silence code. As a result all the silence samples are replaced by a codeword and a sample count.

The above process results in an effective 30 sample delay between input and output samples.

After every 256 sample block of speech is processed, the maximum sample value in the block is determined and stored in a history buffer. This buffer contains the maximum values of 9 previous blocks of speech. The 'silence level' is determined by calculating the average value of the buffer and dividing it by a fixed speech to silence ratio (set at 2,5 by NEERI).

The silence portions are reconstructed by the decoding algorithm by determining the number of samples to be inserted from the silence sample count which follows the silence code. Each of the inserted samples are set to zero volts.

5.5.1 Program Flow Diagram and Variables Used

A full listing of the TMS 32020 source code is given in section 1 of the 'SPEECOM Int[illegible] Software Documentation Manual'.

The data memory of the TMS 32020 is divided into 512 pages each comprising 128 storage locations. Pages 4 through 7 specify 'on-chip' RAM. All other pages are available for use if the appropriate external RAM is provided. The main advantage of using the internal RAM is that the execution speed is optimized in this configuration (for more information consult the 'TMS 32020 Users Guide').

Temporary data storage for this program has been assigned to page 4 of internal memory. The following symbols have been defined in the program code to provide for the symbolic addressing of data memory locations. Each of these locations is used to store the following data:

- HISBAS - the 'physical address' of the base of the History buffer. The base address is represented by the symbol HISTOR.
- HISTAD - a pointer which is used to point to the oldest entry in the History buffer.
- CONLEV - stores the silence level value.
- ABSAMP - the absolute value of each input sample is stored here.
- TOTAL - a variable that contains the sum of all the entries in the History buffer.

SIL - this variable is similar to the 'Boolean' type variable found in Pascal as it can only take on two possible values. The variable is used to indicate whether the input signal is classified as silence or as speech. If SIL = 1 then the speech mode is active and if SIL = Ø then the silence mode is active.

COUNT - this variable is the silence counter. It is incremented if the absolute value of the input sample exceeds the silence level and decremented if this is not true

SILCNT - this variable is used to count the number of silence samples that must be reconstructed when the speech is decoded

MAX - the maximum value of each 256 sample block

The Data buffer and the History buffer are implemented as rotating buffers. These buffers are used in applications where a finite number of data values are to be stored such that each new entry in the buffer is accommodated by discarding the oldest entry (First In Last Out). The classical method of implementing such a buffer is to shift each of the entries one location downwards and placing the latest entry at the top of the buffer (with the oldest entry shifted out of the buffer). This method is cumbersome and consumes an appreciable amount of CPU time (as much as 40 CPU clock cycles could be used to place a new entry in such a buffer).

A rotating buffer solves this problem by keeping the data static and rotating a data pointer in the buffer. This pointer always points to the oldest entry in the buffer. The only operation neccesary to implement such a buffer is a check to ensure that the pointer position does not fall below the base of the buffer. If this condition does occur the buffer pointer is set to the top of the buffer (only 5 clock cycles are used to implement this).

The Data buffer is used to store the previous 30 samples of input data and the History buffer stores the previous 10 history values used in the calculation of the silence level. Extensive use is made of the auxillary registers in the TMS 32020 to implement the Data buffer pointers and general indirect addressing. There are 5 such registers (AR0 - AR4) and operations such as pipelined incrementing and decrementing of these registers are possible. In addition arithmetic capabilities are incorporated in the instruction repertoire associated with the auxillary registers which allow auxillary register 0 to be added to or subtracted from any of the other auxillary registers. The auxillary registers are each 16 bits wide and are thus capable of directly addressing the entire range of data memory. They provide powerful tools for accessing buffers, arrays as well as sequential locations in data memory.

The silence extraction algorithm is presented below:

```
begin
  initialize the history buffer
  initialize the data buffer
  initialize variables
  initialize the history buffer pointers
  set AR2 to 256
  set AR1 to top of data buffer
  set AR0 to base of data buffer
  repeat
    input a sample to @AR1
    set ABSAMP equal to the absolute value @AR1
    if ABSAMP > MAX then
      set MAX equal to ABSAMP
    end if
```

```
if ABSAMP > CONLEV then
  decrement COUNT
  if COUNT < Ø then
    set COUNT equal to Ø
    if SIL = 1 (silence) then
      set SIL to Ø (not silence)
      output the silence code and SILCNT
    end if
  end if
else
  increment COUNT
  if COUNT > 35 then
    set COUNT to  35
    if SIL = Ø (not silence) then
      SIL = 1 (silence)
      set SILCNT to zero
    end if
  end if
end if

--- check the data buffer pointer

if AR1 is at the base of the data buffer then
  set AR1 to top of the buffer
else
  decrement AR1
end if

-- output the oldest value of the data buffer if
-- the speech mode is active

if SIL = Ø then
  output data @AR1
end if
decrement AR2
until AR2 equals Ø
```

```
-- now determine the new silence level

if HISTAD is below HISBAS then
  set HISTAD to top of history buffer
end if
load TOTAL (sum of the contents of the history buffer)
add the value of MAX
set AR2 to HISTAD
subtract @AR2
store in TOTAL
save MAX @AR2
decrement HISTAD
divide TOTAL by 10 (to get average value of buffer)
divide TOTAL by 40 (speech to silence ratio)
store result in CONLEV
set AR2 to 256
set MAX to 0
end.
```

The above algorithm is continuously repeated.

5.5.2 Evaluation of the Program

The above program functioned as required. Program execution at worst case (during the speech mode) was found to be 5,8% of available real time. This percentage of real time is calculated as follows:

$$\% \text{ execution} = \frac{(\text{execution time for X samples})(\text{Fsamp})}{X} . 100$$

where 'Fsamp' is the system sampling frequency.

The results of the silence extraction are shown in Fig 5.1 which shows the original speech waveform, the coded waveform and the reconstructed waveform. The vertical dashed lines indicate embedded silence codes.

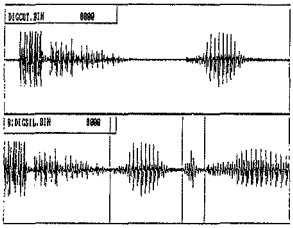

Figure 5.1 Silence coding (a) the original waveform (b) the waveform after extraction. The vertical lines indicated where the silence is extracted

5.6 Segmentation

The segmentation portion of the algorithm performs two functions, namely, the partitioning of the input speech signal into pitch periods or multiples thereof (for voiced segments) and the voiced/unvoiced classification of each segment.

There are numerous algorithms available that extract the pitch information from speech signals. Most of these algorithms rely on complex filtering, auto-correlation and transform techniques which restrict their implementation in real time systems. The method used in this project is a time domain peak picking technique which exhibits fast execution times despite its apparent complexity.

The voiced/unvoiced decision is used both in the extrema coding and short coding portions of the algorithm to select different coding parameters for voiced and unvoiced speech. The decision is based on two criteria, namely, the short time zero crossing rate and the relative maximum amplitude of the segment.

5.6.1 Voiced/Unvoiced Algorithm

The short time zero crossing rate provides a simple measure of the frequency content of the signal. This is particularly true of narrowband signals. The speech signal is essentially a broadband signal, however, the short time zero crossing rate appears to be a useful criterion in making the distinction between voiced and unvoiced signals.

In general, fricatives have a higher zero crossing rate than vowels, however, distibution curves of zero crossings for voiced and unvoiced signals overlap (see Fig 5.2). As a result the decision can not be made using the short time zero crossing alone.

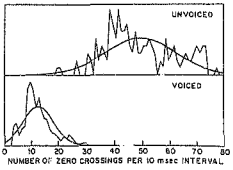

Figure 5.2 Distribution of zero crossings for voiced and unvoiced speech

The second criterion used is the relative maximum amplitude of a segment. It is calculated as follows:

The voiced/unvoiced decision is used both in the extrema coding and short coding portions of the algorithm to select different coding parameters for voiced and unvoiced speech. The decision is based on two criteria, namely, the short time zero crossing rate and the relative maximum amplitude of the segment.

5.6.1 Voiced/Unvoiced Algorithm

The short time zero crossing rate provides a simple measure of the frequency content of the signal. This is particularly true of narrowband signals. The speech signal is essentially a broadband signal, however, the short time zero crossing rate appears to be a useful criterion in the distinction between voiced and unvoiced signals.

In general, fricatives have a higher zero crossing rate than vowels, however, distibution curves of zero crossings for voiced and unvoiced signals overlap (see Fig 5.2). As a result the decision can not be made using the short time zero crossing alone.

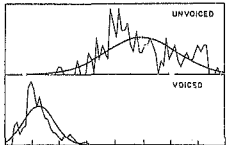

Figure 5.2 Distribution of zero crossings for voiced and unvoiced speech

The second criterion used is the relative maximum amplitude of a segment. It is calculated as follows:

$$\text{Relative Maximum Amplitude} = \frac{\text{Maximum amplitude of segment}}{\text{Conversation level}}$$

The conversation level is calculated by considering the maximum values of 10 previous blocks, each consisting of 1024 samples. These values are stored in a 'maximum amplitude history' buffer and the conversation level is determined by finding the maximum value in the buffer.

The voiced/unvoiced decision is made according to Fig 5.3.

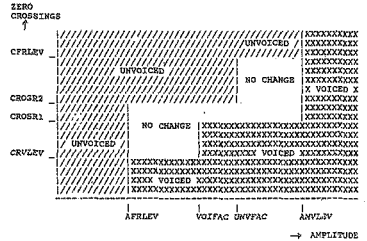

Figure 5.3 Voiced and unvoiced areas as a function of the amplitude level and the number of zero crossings

5.6.2 Pitch Period Determination

The pitch extraction algorithm essentially relies upon the fact that the pitch does not change dramatically in successive pitch periods. Thus if the pitch period is approximated by selecting two successive peaks separated by a distance P then a third peak should be located at approximately a distance P from the second peak (see Fig 5.4). If the magnitude of the third peak is

comparable with the first and second peaks then the initial approximation of the pitch is considered to be correct.

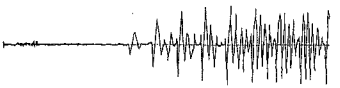

Figure 5.4 Quasi-periodic nature of pitch

The above description forms the basis of the pitch extraction or segmentation algorithm. This algorithm is used to partition the continuous speech waveform into packets or segments. Each of these segments should ideally consist of a number of samples equal to one pitch period for voiced speech and any arbitrary value for unvoiced speech. The purpose of segmentation is to prepare the speech signal data for pitch synchronous analysis, the benefits of which were detailed in section 3.3.

The segmentation algorithm can thus be thought of as the system controller as it places the speech samples into packets and passess these packets to the routines that perform the coding. These routines comprise the background tasks as they operate only on individual segments whereas the foreground tasks of silence extraction and segmentation operate on continuous streams of data (see Fig. 5.5).

The segmentation routine is implemented by using a number of peak picking algorithms termed the Wide, Broad, Narrow, Divide Pitch and Fast Pitch modes. As is evident from the choice of the mode names, their function is to sequentially scan the input waveform with the intention of 'narrowing' down on an estimate of the pitch period. The initial pitch determination is achieved by executing the Wide, Broad, Narrow and Divide Pitch modes. Once these modes have processed the input speech and a successful

pitch determination has been made, the Fast Pitch mode is continuously executed until the pitch changes dramatically or the speech becomes unvoiced.

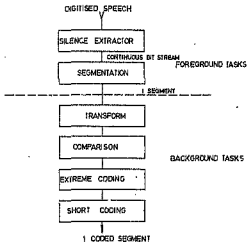

Figure 5.5 Coding algorithm task structure

The process of segmentation begins with the execution of the Wide mode. Its function is to search for the maximum sample value in a window which is defined to be between the 30th and the 200th sample from the origin (see Fig. 5.6(a)). The window parameters have been determined by considering the minimum and maximum pitch periods for male and female speakers when using a sampling frequency of 12,5 kHz.

The search origin is then shifted to the position of the Wide mode peak. The Broad mode then searches for the next peak using the same window length as the Wide mode (see Fig 5.6(b)). The pitch is approximated by the distance between the Broad and Wide mode peaks. This pitch estimate is then tested by the Narrow mode on three successive segments.

The search window for the Narrow mode is centered at the estimated value of pitch and extends to 10% either side of this

value (see Fig 5.6(c)). An estimate of the required magnitude of the Narrow peak, based on the magnitude of the two previous peaks, is made. If the actual peak found falls within certain limits of the estimated peak then the segment is classified as a valid segment. If three successive valid segments occur, the original pitch is accepted.

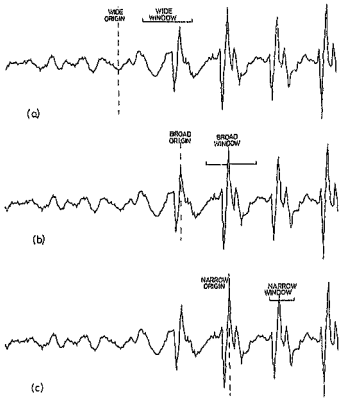

Figure 5.6 Segmentation algorithm (a) Wide mode search (b) Broad mode search (c) Narrow mode search

The pitch estimate at this stage could be a multiple of the actual pitch period. This error can be tolerated although the compression characteristics during the segment comparison portion

could be improved if a single pitch period is used. For this reason a modified form of the Narrow mode (called the Divide Pitch mode) is invoked. This mode divides the estimated pitch by a factor of 2. The new pitch estimate is tested in the same way as the Narrow mode (i.e three successive segments must pass the segment test criterion). If the pitch estimate fails the Divide Pitch test the original pitch is restored otherwise the divided pitch is adopted as the true pitch. The segments determined by the Wide and Broad modes are then reformatted using the determined pitch and 'processed' (i.e. each segment is passed to the background routines for further processing).

The Fast Pitch mode is then initiated. This mode uses the same window length and peak testing algorithm as used in the Narrow mode, the difference being that only one segment is tested and if it passes the peak test it is accepted as a valid segment. This mode is used until a segment fails the peak test. The Wide mode is then invoked and the cycle repeats.

Unvoiced signals are segmented using constant length segments. The segment lengths used are equal to the value of the last voiced segment prior to the detection of the unvoiced condition. This is done so as to provide a smooth transition for voiced to unvoiced. Figure 5.7 shows an example of this.

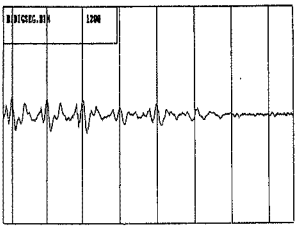

Figure 5.7 Segmentation at transitition from voiced speech

5.6.3 Program Flow and Variable Allocation

The above description has presented a broad overview of the segmentation and voiced/unvoiced decision algorithms. In this section the details associated with the TMS 32020 implementation will be discussed.

The segmentation routine is required to operate on blocks of data samples and, as will be evident in the detailed descriptions given below, many of the algorithms require that a number of previous segments be available for further processing. This implies that a fairly large data buffer be available for the storage of sample data. In addition routines that monitor the status of the buffer should be provided so as to prevent overflow.

To accomplish this the following system for managing the continuous stream of input data has been implemented. A 2048 word data buffer (which is located in external data RAM starting at page 8 ie. address 1024) is used to store the input data. The data is input into the TMS 32020 and stored in successive memory locations in the buffer. This data is then processed and one or more segments are produced. The result of the segmentation routine takes the form of entries in a 'parameter block'. The entries detail the size, location and voiced/unvoiced classification of the segment (or segments) in the data buffer. The data in the parameter block is used by a routine which 'processes' each segment i.e. passes the segment to the background tasks for further processing.

Routines have been incorporated that monitor the status of the buffer. Three data pointers have been defined to manage the buffer. The FILL pointer always points to the first free location in the buffer, the START pointer is used to mark the origin of the current segment and the pointer LAST is used to mark the end of the current segment. After each segment has been determined the START pointer takes on the value of the LAST pointer. Prior

to any processing the pointers are checked to see that sufficient samples are available for processing. If there is insufficient samples, the required samples are added to the buffer from the position specified by FILL. Thereafter the FILL pointer is updated. The algorithm used is as follows:

```
begin
  initiate Wide mode search
  if speech is voiced then
    initiate Broad mode search
    update the parameter block
    if speech is voiced then
      calculate pitch
      repeat
        initiate Narrow search
        test the peak found
      until peak fails test or three Narrow searches complete

      if no failure then
        initiate Divide Pitch test
        if divide pitch pass then
          pitch = pitch / 2
        end if
        reformat the Wide and Broad segments
        update parameter block for all 3 Narrow segments
        process 5 segments
        repeat
          initiate Fast Pitch test
          if passes Fast Pitch test then
            update the parameter block
            process the segment
          end if
        until Fast Pitch fails
      end if
    end if
  end if
  process segments
end.
```

The segmentation algorithm has been implemented with extensive use of subroutines. This approach is motivated by the complexity of the algorithm and the repeated execution of tasks such as buffer filling, voiced/unvoiced classification etc. Each subroutine call consumes two TMS 32020 cycles and when processing individual samples (e.g. silence extraction) this could prove to be time consuming, however, the segmentation algorithm operates on relatively large blocks of data which render the time consumed to implement subroutines as negligible.

The Buffer Fill Procedures

These procedures are responsible for filling the buffer with the number of samples required by the search window. The search process is referenced to the position of the START pointer. The amount of samples available for processing is calculated by subtracting the value of the FILL pointer from the START pointer.

During each of the modes described above, the buffer is filled in two phases. The first phase ensures that the buffer contains a number of samples equal to the current pitch value. These samples are then processed by the classification routine to determine if the speech is voiced or unvoiced. If the speech is determined to be unvoiced the current size of the segment (i.e the previous valid pitch) constitutes the required segment size. The result is that there will be no excess samples in the buffer (the FILL and LAST pointers will be equal) and the processing will be simplified.

If the speech is voiced the second phase of the fill process takes place. The buffer is filled so that the number of samples between the START and FILL pointers is sufficient for the a scan of the entire window (e.g. for the Wide mode there has to be 200 samples available for processing).

In most cases (except after an unvoiced segment) the buffer will contain excess samples i.e. the FILL pointer will be located in a higher position relative to the START or LAST pointer (after each mode is executed the START and LAST pointers are set equal). The fill routines take this possability into account and the number of samples to be added to the buffer is calculated as follows:

No of samples = Required number of samples - Fill + Start

The first phase of the buffer fill routine is implemented by a subroutine called FILLBF. This routine makes use of the TMS 32020 feature that allows for the repetition of a particular command. The number of repetitions can be contained in a data memory location (for the RPT command) or it can be an immediate operand (the RPTK instruction). This instruction is best used in conjunction with an auxillary register operation where the register is incremented or decremented automatically. The FILLBF routine uses the command to input the required amount of samples into the buffer. The program is described as follows:

```
begin
  check if 200 locations are available in the buffer
  if not then
    swop the excess samples
  end if
  calculate the number of samples to be input -> INDCNT
  set AR1 equal to FILL (first free location)
  do for INDCNT times
     input data @AR1 and increment
  end do
  save AR1 in FILL
end.
```

The 'swop' routine is called when there is determined to be insufficient space in the buffer. This routine transfers all the samples between the START and FILL pointers to the base of the buffer (i.e. the samples that have been excluded from the

previous segment). The maximum number of samples that can be transferred is limited to a maximum of 20% of the previous pitch (as the Narrow and Fast Pitch modes have a window which extends 10% either side of the estimated pitch). A large buffer therefore decreases the amount of 'swops'that have to take place.

The second phase of the fill procedure is implemented by a subroutine called FILRST. This routine utilizes a global parameter called RIGHT (to indicate the right hand extreme of the search window) to calculate how many samples need to be added to the existing segment. At the end of this routine the FILL pointer should be located a distance of RIGHT samples from the START pointer.

Buffer Management

In the above sections a number of references have been made in connection with the effective management of the 2048 location data buffer. This management is vital to ensure that signal samples are not lost during execution.

The buffer management procedures can be summarised as follows:

- after every unvoiced segment a buffer swop takes place (the swop procedure was defined in the above section). This swop executes efficiently as there are no excess samples in the buffer (i.e. the LAST and FILL pointers are equal) after processing an unvoiced segment. The pointers START and FILL are set to point to the base of the buffer.
- before the execution of a Wide mode search the buffer is checked to see if there are 1000 free locations. This value is calculated by considering that the maximum number of segments after a Wide, Broad and Narrow search is five and each segment consists of a maximum of 200 samples
- if there is less than 1000 samples available, the buffer is checked to see if there is space for 400 samples. If this is so a variable called NARSWP (Narrow Swop) is set

to 1. This action means that the samples used for the Wide and Broad modes will be located near the top of the buffer, however, as soon as the Narrow mode is invoked a 'swop' takes place and the Narrow segments are processed at the base of the buffer. The NARSWP condition prevents the execution of the reformatting operations that are usually performed on the Wide and Broad segments after the correct pitch is determined

The Parameter Block

A parameter block is used to store the information concerning the position in the data buffer and classification of each segment. This block effectively contains the results of the segmentation procedure. The block is located in page 4 of the data storage space (the symbol SEGDAT points to the base of the parameter block).

A subroutine called UDTSEG is used to update the parameter block with the segment parameters. A pointer called SEGPTR is used to point to the first free location in the parameter block. The routine stores firstly the classification followed by the value of the pointer LAST and finally the value of the START pointer. These parameters are used by the subroutine called PROCES to output the data values to a 'port'. The routine PROCES also clears the parameter block and sets the value of SEGPTR equal to SEGDAT i.e. the base of the parameter block.

The Voiced/Unvoiced Classification

This classification always takes place directly after the first phase of the buffer fill process. Two subroutines called CLASFY and CHKBND perform the classification tasks. CLASFY determines the zero crossing rate and the maximum value of the current segment i.e. between the START and FILL pointers. The CHKBND subroutine checks these parameters to determine if the speech is voiced or unvoiced.

The algorithm used in the routine CLASFY is as follows:

```
begin
  AR1 = START
  AR2 = PITCH-3
  AR0 = 3
  AR3 = 0
  repeat
    load data @AR1
    if data > WINMAX then
      WINMAX = data
    load data @AR1 and increment
    if data < 0 then
      load data @ AR1
      if data > 0 then
        increment AR3
      end if
    else
      load data @AR1
      if data < 0 then
        increment AR3
      end if
    end if
    decrement AR2
  until AR2 = 0
  CRSRAT = AR3
  CRSRAT = CRSRAT*100/PITCH
end.
```

The routine calculates two parameters, namely, the maximum sample value between the Start and Fill pointers and the zero crossing rate CRSRAT.

The fractional arithmetic used in the CHKBND routine is accomplished by specifying the fractions as Q6 integers (for information concerning the 'Q' notation see TMS 32020 users guide). This particular value was chosen so as to use the TMS

J2020 feature that allows for the automatic shifting of results of a multiply operation six places to the right. The result is that a Q6 number can be multiplied by a Q0 (integer) number with the answer appearing in the accumulator as a Q0 number.

Finding the Peak Value

A subroutine called FINMAX is used to search for the peak sample value in the search window. The search is conducted between the two parameters that specify the window length. A parameter called LEFT represents the start position of the window search and the parameter RIGHT represents the end of the search window. The physical addresses of the first and last samples in the search are calculated by adding the values LEFT and RIGHT to the segment origin START.

The subroutine determines the peak sample in the search window. Its value and position (relative to START) are stored in the variables MAX and LAST, respectively. The value WINMAX, which was computed during the voiced/unvoiced classification routine CLASFY is updated if the value of MAX exceeds it. The value WINMAX thus contains the peak value in the segment (ie. between the pointers START and FILL) and the variable MAX contains the peak value in the search window.

The value WINMAX is passed to the routine used to update the conversation level.

The algorithm used in the routine FINMAX is as follows:

```
begin
  LEFT = START + LEFT
  RIGHT= START + RIGHT
  INDCNT = RIGHT - LEFT - 1 (no. of samples to be searched)
  AR2 = INDCNT
  AR1 = LEFT
  MAX = -32000
  repeat
    load data @AR1 and increment
    if data > MAX then
      LAST = AR1
      MAX = data
    end if
    decrement AR2
  until AR2 = 0
  if MAX > WINMAX then
    WINMAX = MAX
  end if
  update the conversation level
end.
```

Updating the Conversation Level

The subroutine UPDCNV is used to perform the calculations which result in the conversation level being updated every 1024 samples of voiced speech. The subroutine is executed as follows:

```
begin
  if FILCNT > 0 then
    if WINMAX > VOIMAX then
      VOIMAX = WINMAX
    end if
    SMPCNT = FILCNT + SMPCNT
    if SMPCNT > 1024 then
      AR1 = HISPTR
      AR0 = base of the History Buffer
      store VOIMAX @AR1 and decrement
      HISPTR = AR2
      set CNVLEV, VOIMAX and SMPCNT to 0
      repeat
        load data @AR2 and decrement
        if data > CNVLEV then
          CNVLEV = data
        end if
      until AR2 < AR0
      if HISPTR < AR0 then
        HISPTR = top of history buffer
      end if
    end if
  end if
end.
```

The variable FILCNT represents the number of voiced samples added to the buffer by the FILBUF and FILRST routines. The variable VOIMAX contains the maximum sample value in the current 1024 sample block.

The UPDCNV routine utilizes the auxillary register compare statement 'CMPR' which tests the value of the currently selected auxillary register by comparing it with the contents of auxillary register 0. The type of comparison i.e. equal to, greater than or less than, is dependent on the immediate parameter supplied on the CMPR command (e.g. CMPR 0 tests if the auxillary register is equals AR0). The results of the comparison are tested by using the

BBZ (Branch if bit is zero) o. BBNZ (branch if bit is not zero). The 'bit' is set if the tested condition is true.

In UPDCNV register AR0 is set to equal the base address of the history buffer. The CMPR command is used to determine when register AR2 and the history buffer pointer HISPTR are below the base of the history buffer.

Peak Testing and Predicting

During the Narrow, Fast Pitch and Divide Pitch search modes a prediction is made of the expected value of the segment peak. The actual peak value is then tested to see if it falls within specified limits of the estimated peak. If the peak passes the test the segment is accepted.

A subroutine called PRDICT is used to estimate the peak value of the next peak based on the values of the two preceding peaks. The PRDICT algorithm functions as follows:

```
begin
  SLOPE = OLDMAX - MAX
  if SLOPE < 0 then
    ESTMAX = MAX
  else
    ESTMAX = MAX - SLOPE
  end if
  OLDMAX = MAX
end.
```

The variables OLDMAX and MAX are the values of the two previous peaks (with MAX representing the last peak). ESTMAX contains the estimated value of the next peak.

The routine TSTSEG is used to determine if the segment is accepted as a valid representation of a pitch period. The segment test algorithm functions as follows:

```
begin
  if speech is voiced then
    if MAX > Ø then
      TEMP1=WINMAX * (1/SEGRAT)
      if MAX > TEMP1 or OLDMAX > TEMP1 then
        PEAKCT = Ø
        if ESTMAX*RATIO > MAX and MAX*RATIO > ESTMAX then
          segment is good
        else
          segment is not good
        end if
      else
        PEAKCT = PEAKCT + 1
        if PEAKCT > 2 then
          segment is not good
        end if
      end if
    else
      segment is not good
    end if
  else
    segment is good
  end if
end.
```

The above routine implements all fractional arithmetic in Q6 fixed point notation. This notation is once again used to take advantage of the automatic six bit right shift available at the output of the multiplier.

A variable called PEAKCT is used to store the extra peak detect counter. This counter is incremented everytime the peak value of WINMAX (multiplied by a scaling factor of Ø,66) exceeds the values of OLDMAX and MAX. This condition implies that the peak search procedures have missed the true pitch peaks. If this condition occurs twice in succession the segment fails the peak test.

```
begin
  if speech is voiced then
    if MAX > 0 then
      TEMP1=WINMAX * (1/SEGRAT)
      if MAX > TEMP1 or OLDMAX > TEMP1 then
        PEAKCT = 0
        if ESTMAX*RATIO > MAX and MAX*RATIO > ESTMAX then
          segment is good
        else
          segment is not good
        end if
      else
        PEAKCT = PEAKCT + 1
        if PEAKCT > 2 then
          segment is not good
        end if
      end if
    else
      segment is not good
    end if
  else
    segment is good
  end if
end.
```

The above routine implements all fractional arithmetic in Q6 fixed point notation. This notation is once again used to take advantage of the automatic six bit right shift available at the output of the multiplier.

A variable called PEAKCT is used to store the extra peak detect counter. This counter is incremented everytime the peak value of WINMAX (multiplied by a scaling factor of 0,66) exceeds the values of OLDMAX and MAX. This condition implies that the peak search procedures have missed the true pitch peaks. If this condition occurs twice in succession the segment fails the peak test.

The result of the peak test routine is represented by a value of 1 being stored in the variable GOOD if the test was successful and a value of 0 if the test fails.

The Wide Mode

The Wide mode algorithm is given as follows:

```
begin
  fill the buffer
  classify the segment
  if speech is voiced then
    LEFT = 32
    RIGHT = 200
    fill the rest of the segment
    find the maximum in the window
    OLDMAX = MAX
  else
    LAST = FILL
  end if
end.
```

The peak search is in the window from START+32 to START+200. The maximum found is stored in MAX and OLDMAX.

If the speech is unvoiced then the LAST pointer is assigned to the position of the FILL pointer whereas for voiced speech the LAST pointer points to the position of the Wide mode peak. The values of the START and LAST pointers are stored in the parameter block by the 'update parameter block' routines.

The Broad Mode

The Broad mode algorithm functions as follows:

```
begin
  fill the buffer
  classify the segment
  if speech is voiced then
    LEFT = 32
    RIGHT = 200
    fill the rest of the segment
    find the maximum value in the window
    BRDMAX = MAX
    BRDEND = LAST
    PITCH = LAST - START
    predict the next peak
  else
    LAST = FILL
  end if
end.
```

The Broad mode algorithm is similar to the Wide mode. The main difference between the modes is that the Broad mode makes the first approximation of the pitch period. This value is stored in the variable PITCH. The variables BRDMAX and BRDEND are used to store the Broad mode peak parameters MAX and LAST, respectively. This operation is necessary for two reasons, firstly, if the Narrow search fails to confirm the pitch estimate the value of BRDEND is used to intialize the next search origin to the end of the Broad segment. Secondly, the value of BRDMAX is used to restore the value of OLDMAX for the Divide Pitch mode.

The peak prediction routine is called by the Broad mode algorithm to provide the estimated peak value for the first Narrow mode segment.

The Narrow Mode

The Narrow mode is used to confirm the estimated value of pitch as supplied by the Broad mode. The Narrow mode algorithm is as follows:

```
begin
  TEMP2 = 0
  if NARSWP=1 then
    swop the samples
    set BRDEND to the base of the buffer
  end if
  repeat
    fill the buffer
    classify the segment
    if the speech is voiced then
      calculate the search window
      fill the rest of the segment
      find the maximum value
      if TEMP2 = 0 then
        SLOPE = MAX - OLDMAX
      end if
      PITCH = LAST - ST
      test the peak found
      if the peak is good then
        update the parameter block
        START = LAST
      else
        SEGPTR = SEGPTR - TEMP2*3
        START = BRDEND
      end if
      predict the next peak
      NARMAX = OLDMAX
      NAREST = ESTMAX
      TEMP2 = TEMP2 + 1
    else
      segment is not good
      LAST = FILL
      update the parameter block
    end if
  until a segment is not good or TEMP2 = 3
end.
```

The Narrow mode performs three consecutive tests to determine if the pitch estimate supplied by the Broad mode is correct. If any one of the 3 tests fails, the search origin is re-positioned at the end of the previous Broad mode and the Wide search mode is invoked.

A routine is provided to calculate the required search window parameters (i.e. the values of LEFT and RIGHT) for the Narrow (as well as Divide Pitch and Fast Pitch) mode. This routine determines the search window to be +/- 10 % either side of the estimated (or previous) pitch. This allows for short time variations of the pitch period.

A number of new variables appear in the Narrow mode algorithm these variables and there respective functions are summarized below:

SLOPE - this variable is used to store the slope of the line that is drawn between the Broad mode peak and the first Narrow peak. The purpose of SLOPE is to provide the estimated peak value of the first segment in the subsequent Divide Pitch mode test (see Fig 5.8).

TEMP2 - this variable functions as the segment counter. Once three segments have been successfully tested the Narrow mode is terminated.

NARMAX- the peak value found in the final segment tested in the Narrow mode is stored in this variable when the mode is completed. This value is used to restore the value of OLDMAX if the Divide Pitch tests fail.

NAREST- the next estimated peak is stored in this variable. As with NARMAX the value of ESTMAX is restored if the divide pitch test fail.

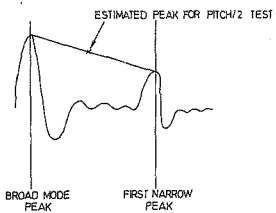

Figure 5.8 Calculation of peak estimate

In the event that one of the Narrow segments fails the test the parameter block pointer SEGPTR is adjusted so that all the Narrow segments are discarded from the parameter block.

The Fast Pitch Mode

This mode is similar to the Narrow mode. The purpose of this mode is to take advantage of the fact that once the pitch of a voiced portion of the speech signal is found, there should be little relative change between succeeding segments. The pitch can thus easily be determined and tested. The Fast Pitch algorithm implemented by the routine FSTPCH and is as follows:

```
begin
  fill the buffer
  classify the segment
  if speech is voiced then
    calculate the search window
    fill the rest of the segment
    find the maximum in the window
    test the peak found
    if the segment is good then
      update the parameter block
      process the segment
      PITCH = LAST - START
      LAST = START
      OLDPCH = PITCH
      predict the next peak
    end if
  else
    segment is not good
    LAST = FILL
    update the parameter block
  end if
end.
```

The Fast Pitch mode is repeatedly executed until a segment fails the peak test or is classified as unvoiced.

The variable OLDPCH is used to store the last valid pitch value. This value is used to segment the unvoiced portions of a signal when a voiced signal changes to unvoiced.

The Divide Pitch Mode

This mode is used to test if the pitch determined by the Broad and Narrow modes is actually double the actual pitch. The subroutine DIVPCH is used to implement the mode. The algorithm is as follows:

```
begin
  START = BRDEND
  OLDMAX = BRDMAX
  ESTMAX = OLDMAX + SLOPE/2
  TMPPCH = PITCH
  PITCH = PITCH / 2
  TEMP2 = 1
  AR4 = base of the temporary parameter block (TMPDAT)
  repeat
    calculate the search window
    find the maximum in the window
    PITCH = LAST -START
    store voiced code @AR4 and increment
    store LAST @AR4 and increment
    store START @AR4 and increment
    test the peak value found
    predict the next peak
    START = LAST
    if peak is not good then
      PITCH = TMPPCH
      START = NAREND
      ESTMAX = NAREST
      OLDMAX = NARMAX
    end if
    TEMP2 = TEMP2 + 1
  until TEMP2 = 4 or peak is not good
  if peak is good then
    remove the Narrow segment params. from the param. block
    transfer the temporary parameter block
  end if
end.
```

Auxillary register AR4 is used as a pointer for the temporary parameter block. This block is used to store the parameters of the three segments tested in the divide pitch mode. If the divide pitch test passes then the Narrow mode portions of the parameter block are replaced by the data in the temporary parameter block.

Reformatting the Segments

There remain two subroutines still to be discussed. Both of these routines have the task of changing the segments generated by preceding modes. The first is the routine BCKSTP (Backstep) which is used to reformat the wide and broad segments after the true pitch has been determined. The BCKSTP algorithm is as follows:

```
begin
  TEMP1 = START
  TMPPTR = top of the temporary buffer
  AR1 = TEMPPTR
  AR2 = SEGPTR
  AR3 = 8
  decrement AR2 to point to last entry in param. block
  repeat
    load data @AR2 and decrement
    store data @AR1 and decrement
    decrement AR3
  until AR3 = 0
  TEMP2 = BRDEND
  while TEMP2 - start of the Wide seg > PITCH + 32 do
    TEMP2 = TEMP2 - PITCH
  end while
  initialize SEGPTR to the base of the parameter block
  START = start of the Wide segment
  LAST = TEMP2
  update the parameter block
  while TEMP2 <> BRDEND do
    START = TEMP2
    LAST = START + PITCH
    TEMP2 = LAST
  end while
  AR1 = SEGPTR
  AR2 = base of temporary data block
  AR3 = 8
```

```
    repeat
      load data @AR2 and increment
      store data @AR1
      decrement AR3
    until AR3 = Ø
    SEGPTR = AR1
    START = TEMP1
  end.
```

The above algorithm backsteps from the end of the Broad mode in steps equal to the pitch period. This process continues until the number of samples between the start of the Wide segment and the backstep pointer (TEMP2) is between 32 and PITCH+32 points. The parameter block is then updated as the pointer TEMP2 retraces its steps to the start of the Narrow mode. The Narrow mode parameters are temporarily stored at the start of the subroutine and are restored at its conclusion.

The second formatting routine is the unvoiced reformat routine called DTCTUN (Detect Unvoice). The function of this routine is to reformat the segments in the parameter block so as to provide a smooth transition between voiced and unvoiced segments. The routine is thus invoked as soon as the voiced to unvoiced transition occurs. The DTCTUN description is as follows:

```
  begin
    START = start of the Wide segment
    PITCH = OLDPCH
    TEMP1 = LAST
    if LAST > START then
      initialize SEGPTR to base of the param. block
      while TEMP1 - START > PITCH do
        LAST = START + PITCH
        update the param. block
        START = LAST
      end while
    end if
  end.
```

The routine goes back to the start of the first Wide mode. The data in the buffer is then segmented with each segment being of length eq to the last valid pitch.

This routin is called by the processing routine.

5.6.4 Analysis of the Segmentation Routine

The typical results of the segmentation routine are shown in Fig. 5.9. The algorithm has been found to function according to specifications. The results are that the speech signal is correctly segmented during the voiced portions of the signal, however, errors do occur during the transition from unvoiced or silence to voiced speech. These errors will not affect the quality of the speech but the compression statistics will be adversely affected. The contribution of these errors is on the whole negligible and are thus acceptable.

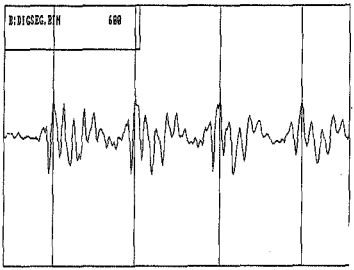

Figure 5.9 Results of segmentation routine

The following results were obtained from the real time analysis of the segmentation procedure:

- the worst case execution was found when the following modes were executed; W B N1 N2 N3(failure). This is evident as only the segments generated by the Wide and Broad modes are sent to the processing routine. As a result the processing involved in computing the Narrow mode tests is effectively wasted. The execution time for the above event was found to be 15% at worst case
- execution of the following modes; W B N1 N2 N3 and successful Divide Pitch resulted in a worst case execution time of 8%
- the Fast Pitch mode was found to consume 4% of real time
- the unvoiced segmentation required 3,5% of real time
- a successful W B N1 N2 N3 search with a Divide Pitch failure used 4% of real time at worst case

The sequence of events that generated the worst case execution have a very low probability of occurring and thus do not contribute significantly to the average execution time of the algorithm (which was found to be 6,3% when tested on one speech file), however, for the purposes of analyzing the real time performance of the whole algorithm it will be necessary to consider the worst case execution times of each portion of the algorithm.

5.7 The Fourier Transform

The function of this portion of the algorithm is to transform each speech segment into the frequency domain. The NERRI simulation routines implemented this portion using the Discrete Fourier Transform. This approach results in a flexible and simple approach to the problem, however, for the purposes of real time implementation it is unacceptable.

As a result a transform algorithm had to be found which satisfied the following requirements:

- it should be able to transform segments consisting of random numbers of data points
- the range of segment sizes should extend from 30 to 180 samples
- the algorithm should execute efficiently in real time

A number of algorithms are available that implement the Fast Fourier Transform. The classic Cooley-Tukey Radix-2 and Radix-4 algorithms provide fast and simple means of executing the Fourier Transform, however, they are limited by the number of transform sizes available in the required interval.

Singleton has developed an FFT program for arbitrary mixed radices. This algorithm gives a dense coverage of segment sizes by representing the transform size as powers of 2, 3 and 5. In 1975 Dr. Winograd of IBM proposed a theory for efficient calculation of prime-length cyclic convolution using a minimum number of multiplications. These results were important as they could be combined with Rader's conversion of the DFT to convolution to give very efficient short prime length DFT algorithms. Winograds algorithm for calculating long transforms, known as the Winograd Fourier Transform Algorithm (WFTA), uses substantially fewer multiplications than the traditional Cooley-Tukey FFT, but at the expense of more additions.

The Prime Factor Algorithm (PFA) calculates the DFT using the prime factor index map of Good and Thomas [Good ,1971], with the resulting short DFT's converted into convolution using Rader's methods and the convolution evaluated by Winograd-type minimum multiplication routines. The theory of this approach was first presented by Kolba and Parks who obtained encouraging results.

A comparison of the execution speeds of the above algorithm was performed by Burrus et al [1981]. The results of these tests are shown in Table 5.1.

TABLE 5.1

The time in Milliseconds to calculate a length N DFT

Length N	PFA	Radix-4	WFTA	Singleton
63	37,6		63,9	84,3
64		43,0		53,8
252	210,7		276,6	396,0
256		229,4		263,3
1008	1002,8		1469,0	1730,0
1024		1129,6		1262,6

The above benchmarks were obtained by testing the algorithms on a PDP 11/45 with 128K of memory and floating point hardware. The floating point hardware had, as is the case in the TMS 32020, approximately equal multiply and add times which makes the benchmark relevant for this project.

Clearly the PFA method is the fastest FFT technique. The transform size coverage of the PFA is dependent on the number of prime factor WFTA modules incorporated in the algorithm. The composite PFA along with the WFTA algorithms for the prime factor modules 2, 3, 4, 5, 7, 8, 9 and 16 are presented in the paper by Burrus et al [1981]. The range of values covered by the decomposition into the above prime factors is given in Table 5.2.

TABLE 5.2

The range of values of the PFA using the WFTA prime factor modules 2, 3, 4, 5, 7, 8, 9 and 16

Length N	Prime1	Prime2	Prime3
30	2	3	5
35	1	7	5
36	4	9	1
40	8	1	5
42	2	3	7
45	1	9	5
48	16	3	1
56	8	1	7
60	4	3	5
63	9	7	1
70	2	7	5
72	8	9	1
80	16	1	5
84	4	3	7
90	2	9	5
105	3	7	5
112	16	7	1
120	8	3	5
126	2	9	7
140	4	7	5
144	16	9	1
168	8	3	7
180	4	9	5

The PFA algorithm satisfies the requirements of execution speed and allows for the transformation of a number of segment sizes in the desired interval. The only feature of the algorithm that presented a problem was its complexity. This was not considered to be a decisive factor as the TMS32020 can accommodate 64K of program memory and the transform algorithms are very well

documented.

The transform portion of the algorithm consists of a number of other algorithms and programs that perform the tasks of processing the segment data for use by the transform algorithm and for computing the magnitude spectrum of the segment from the real and imaginary data values.

The general transform portion executes as follows:

```
begin
  determine closest available transform size
  adjust the segment samples to match the transform length
  perform the PFA FFT
  calculate the magnitude from the real and imaginary parts
end.
```

The additional tasks of segment size adjustment and magnitude spectrum computation present further implementation problems which will be detailed in sections 5.7.2 and 5.7.4, respectively.

5.7.1 Implementation of the Transform Portion

The detailed description of the transform portion of the algorithm is as follows:

```
begin
  initialise the processor
  read the prime factor and unscrambling table into RAM
  input the data points into memory
  determine from lookup table the closest transform size
  adjust the segment length to match the transform size
  set the imaginary values to zero
  perform the PFA
  calculate the magnitude spectrum
  output half the points
end.
```

The program begins by initializing the TMS32020 internal variables. The contents of Table 5.2 together with the 'unscrambling' values (see PFA documentation Burrus [1981]) for each transform size are then transferred from program memory to data memory. This step is necessary as the Harvard architecture of the TMS 32020 does not allow manipulation of data in the program address space. The table is stored in external data memory beginning at address 500H. The following parameters are stored for each transform size:

transform size
prime 1
prime 2
prime 3
unscrambling value

In addition to the above transform size and unscrambling table a lookup table containing the addresses of each of the WFTA transform routines is transferred to data memory starting at location 5A0H. This data is used by the PFA algorithm to locate the WFTA routines.

The segment data values are read into internal data RAM beginning at location 400H. The segment code is stored in CODE and the number of points in the segment in POINTS.

The transform length table is scanned to determine the transform size closest to the value of POINTS. If the new transform size is different to the previous segment transform size (stored in OLDI) and if the difference between their respective segment sizes is less than three samples, the transform size of the previous segment is used. This scheme provides additional segments for comparison in the next portion of the algorithm (see section 5.8).

5.7.2 Segment Length Adjustment

There are two methods that can be used to change the number of samples in the segment to match the selected transform size. The first method is zero-padding. This method is the most simple and the most obvious, however, it is limited by the fact that the transform size has to be chosen to be the next available transform length above the segment size i.e. for a segment size of 145 samples the transform size would be 168 points (see Table 5.2). Over and above this problem zero-padding distorts the amplitude spectrum considerably. This could have disastrous consequences on the eventual speech quality.

The second method uses interpolation or decimation to extend or compress the segment length. This method is advantageous as the transform size closest to the segment length can be used and as a result the amplitude spectrum is less distorted. Digital filtering has been proposed as a means of implementing interpolation and decimation. This method is the logical technique as the TMS 32020 has been designed to implement digital filtering efficiently, however, the filter parameters for each segment size and transform size have to be predetermined and stored in memory. This renders the technique impractical.

The proposed method makes use of resampling by means of linear interpolation. This method assumes that all points between two known data points fall on a straight line drawn between the two points. All the data values on this line can be calculated by using the simple equation

$$Y = MX + C$$

If the first and second points are given by (X1,Y1) and (X2,Y2), respectively then;

$$M = (Y2-Y1)/(X1-X2)$$

and

C = Y1

In Fig. 5.10 the true spectrum (calculated with the DFT) is compared with the results obtained using both the the linear interpolation method and the zero-padding. The linear interpolation preserves the spectrum shape as opposed to zero padding where distortion is evident.

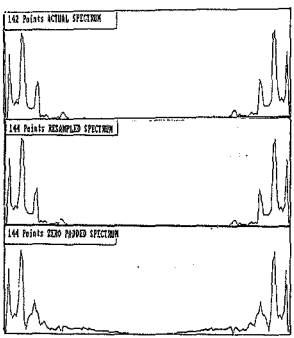

Figure 5.10 Comparison of Actual spectrum with spectrum attained by resampling and zero padding

The complete high level simulation algorithm for the resampling process is given below:

```
begin
  STEP := POINTS / NEW_N
  DIST := 1
  X [1] := T [1]
  for I := 2 to NEW_N do
    begin
      DIST := DIST + STEP
      DIFF := T[TRUNC(DIST)+1] - T[TRUNC(DIST)]
      X[I] := T[TRUNC(DIST)] + (DIFF*FRAC(DIST))
    end
end
```

The variable POINTS refers to the old segment size and NEW_N the transform size. The input data is assumed to be present in buffer T and the resampled segment is stored in X. The function TRUNC provides the integer part of a real number and FRAC the fractional part.

The implementation of the above algorithm on the TMS32020 is complex as the device was not designed to efficiently execute such algorithms. An attempt was made, however, to utilize the TMS32020's special features to improve the execution time of the resampling algorithm.

The program description of the TMS32020 implementation of the resampling algorithm is given below

```
begin
  STEP = POINTS / NEWN
  TEMP2 = NEWN - 1
  set AR3 = TEMP2
  set AR2 = destination data address
  set AR4 = source data address
  set AR0 = 2
  transfer data @AR4 to location @AR2
  DIST = STEP
  repeat
    load DIST
    add the source address
    INT = integer value of accumulator
    FRAC = fractional value of the accumulator
    set AR4 to INT
    load data @AR4 and increment
    subtract data @AR4 and decrement
    negate the accumulator
    store in TEMP1
    multiply with FRAC
    add data @AR4
    store data @AR2 and increment by AR0
    DIST = DIST + STEP
    decrement AR3
  until AR3 = 0
end
```

The most complex part of the above software is the method of splitting the value of (DIST + Base address) into integer and fractional parts. This was accomplished by specifying the variable STEP as a Q7 number. As a result the value of DIST was also Q7, however, when DIST is loaded into the accumulator it is shifted 9 places to the left. Thus the integer part is in the high order word of the accumulator and the fractional part in the low order word. The TMS32020 allows for separate manipulation of the high and low order words of the accumulator and by careful use of the barrel shifter and various masking operations, the

required processing is accomplished.

The resampled segment is stored in internal RAM starting at address 200H.

5.7.3 The PFA Algorithm Implementation

The PFA algorithm used is an in-place algorithm and as a result the transformed data samples are restored in the same position as time domain samples. They are, however, not in-order. The 'unscrambling' constant is used to re-order the samples.

In the implementation the WFTA transform modules for sizes 2, 3, 4, 5, 7, 8 and 9 have been translated into TMS32020 code. Each of these modules makes extensive use of temporary storage locations. These locations must exist in internal RAM as the program execution time increases dramatically if external RAM is used.

5.7.4 The Square Root Algorithm

The final problem area associated with the Transform portion of the algorithm is the requirement that the real and imaginary components of the spectrum be combined to form the magnitude spectrum. This trivial computation is easily accomplished with the simple formula

$$\text{Magnitude} = \sqrt{\text{real}^2 + \text{imaginary}^2}$$

Due to the square root function the above formula is difficult to implement on any micro-processor and the TMS32020 is no different. A number of algorithms which approximate the square root were considered.

The method eventually chosen is a two line linear approximation algorithm [Adams and Brady, 1983] which provides approximations with peak errors less than 1/2 dB. Two line methods generally require more execution time and are more difficult to implement

than simple one line linear approximations. However, a number of special features of the TMS32020 have been utilized which result in efficient execution of the algorithm. The Adams algorithm is given below.

Consider a complex number with components I and Q

```
begin
  X = largest value of I and Q
  Y = smallest value of I and Q
  compare X to Y
  if X < 4Y then
    answer = 7X/8 + Y/2
  else
    answer = X
  end if
end
```

The above algorithm is simple to implement. The calculation of the 7X/8 + Y/2 formula was accomplished as follows: the value of Y was loaded into the accumulator with a 15 bit left shift. This meant that with reference to the high word of the accumulator the value of Y was divided by 2 (a right shift of one bit!). The value of X was then multiplied by the value 7 expressed as a Q12 number. The result of any multiplication is placed in the P register. The P register shifter on the multiplier unit was configured to shift the result of the P register once to the left (by using the command SPM 1). As a result the contents of the P register were in Q13 format.

Q13 format is in fact a 3 bit shift to the right with respect to the high order word of the accumulator and thus when the contents of the P register were added to the accumulator the result was 7X/8 + Y/2 in the high order word (ie Q16 format).

Thus by careful use of a number of the features of the TMS 32020 a possible time consuming and expensive task was performed using 5 instructions e.g.

```
LAC     Y,15                load Y shifted left 15 places
LT      X                   load X into the T register
MPY     (7 IN Q12 FORMAT)   Mult X (in Q0) with 7 (in Q12)
APAC                        add P to Acc (with shift)
SACH    ANSWER              store the high order word
```

As a conclusion to this portion of the algorithm the data samples were unscrambled and together with the segment code and a count of the original number of samples is output via an appropriate port (using the simulator the data was thus stored on disc). It should be noted that only half the data samples are stored due to the symmetry of the spectrum.

5.7.5 Analysis of the Results

The algorithm is dependent on the requirement that the temporary data storage used by the WFTA modules as well as the segment sample buffers (for both resampled and original segments) be located in internal data memory. This improves execution time by approximately 20% of real time. This is a significant and very necessary improvement.

At the present point there is sufficient internal RAM to accommodate the above data, however, the addition of further WFTA modules would definitely result in insufficient temporary data storage in internal RAM. This would have a disastrous effect on the real time performance.

The overall algorithm produced favourable results and a worst case execution time of 30% of real time.

5.8 Segment Comparison

The function of this portion is to compare each segment with its predecessor so as to determine if there is any similarity. If the distortion between the two segments is less than a predetermined value a repeat code is stored in place of the latter segment. The repeat code instructs the decoding algorithm to regenerate the segment by repeating the previous segment.

This portion of the algorithm contributes significantly to the overall compression statistics.

Segment comparison takes place between two segments known as the reference segment and the object segment. Once the comparison takes place, and if the object segment is similar to the reference segment, it is output and renamed as the reference segment. If a comparison results in a repitition code being output the original reference segment is preserved for the next segment comparison.

Segment comparison can only occur if two criteria are satisfied. Firstly, both the reference and object segments both have to be the same voiced type (i.e. voiced or unvoiced) and secondly, they both have to contain the same number of samples.

Once these requirements have been satisfied the relative distortion of the two segments is calculated. If this figure is better than -15dB for voiced segments and -10dB for unvoiced segments, then a segment repetition code is output.

A maximum of 3 consecutive repetitions is allowed as a clearly audible echo effect results from excessive segment repetition.

The general segment comparison routine is given below:

```
begin
  load the first segment
  store without change
  this segment is now the reference segment
  repeat
    load the object segment
    if the voiced/unvoiced codes of obj and ref are equal
    and if the ref and obj segments have the same no. of
    points then
         calculate the distortion
         if voiced and distortion > -15dB  and less than 3
         repetitions have occurred then
           repeat the segment
         else if unvoiced and distortion > -10dB and less
         than 3 repetitions have occurred then
           repeat the segment
         else
           output the obj segment
           set the obj segment to be the ref segment
    else
         output the obj segment
         set the obj segment to be the ref segment
  until the end of the data
end.
```

The distortion is calculated as follows:

$$\text{DISTORTION} = \frac{\Sigma\ (Y-X)^2}{\Sigma\ Y^2}$$

where X = data samples of the reference segment
Y = data samples of the object segment

5.8.1 The Implementation

In the TMS 32020 implementation of the segment comparison algorithm two data pointers called OBJ and REF are assigned to

point to the start of the object and reference segments respectively. The OBJ pointer is initialised to 200H and the REF pointer is set to 2B5H. The use of pointers allows the current object segment to replace the reference segment merely by assigning the value of OBJ to REF. A subroutine called LDOBJ is used to input the data samples into internal RAM begining at the location pointed to by OBJ.

The subroutine CALDST is used to calculate the distortion between the reference and object segments. The sum of squares is the basic function required for this routine. The TMS32020 provides such a facility with the SQRA (square and accumulate) command. The routine is implemented as follows :

```
begin
  set AR1 = REF
  set AR2 = OBJ
  repeat for all the data points
     square data *AR1 and accumulate and increment AR1
  store the high and low words of the accumulator in SUM1
  repeat for all the data points
     square data @AR2 and accumulate and increment AR2
  store the high and low words of the accumulator in SUM2
  SUM2 = SUM1 + SUM2
  set AR1 = REF
  set AR2 = OBJ
  set AR3 = no. of data points
  zero the accumulator
  repeat
    multiply data @AR1 with data @AR2 increment AR1 and AR2
    subtract the P register from the accumulator
    decrement AR3
  until AR3 = 0
  shift accumulator left ie multiply by 2

  -- know have -2XY in the accumulator
```

```
    add SUM2 to accumulator and store back in SUM2
    if SUM2 > SUM1 then
      normalize SUM2 and store the high order word
      adjust SUM1 accordingly
    else
      normalize SUM1 and store the high order word
      adjust SUM2 accordingly
    end if
    divide SUM2 by SUM1 to give distortion DIST
  end
```

A number of subtle points should be noted in connection with the above routine. Firstly, the formula

$$\frac{\Sigma (X-Y)^2}{\Sigma Y^2}$$

is multiplied out to yield the following equation

$$\frac{\Sigma X^2 - 2XY + Y^2}{\Sigma Y^2}$$

this equation is then used in the calculation of the segment distortion.

Secondly, the variables SUM1 and SUM2 are 32 bit numbers and are thus divided into low and high order words ie. SUM1L and SUM1H and similarly for SUM2. The 32 bit arithmetic is necessary to prevent overflow during the square and accumulate computations. This, however, introduces a problem as the TMS32020 is only capable of executing division of 16 bit numbers efficiently. This problem is overcome by comparing the two 32 bit numbers SUM1 and SUM2. The larger number is then normalized (it is shifted left until the most significant bit is a 1) and the smaller number is shifted accordingly. The high order words are then used for the division. This method preserves adequate accuracy for the distortion calculation.

5.8.2 Analysis of the Results

The above algorithm was found to execute very efficiently. An execution time of 1,48% of real time was found to be the worst case duration. This time included reading and writing of the input samples from an external device.

5.9 Extrema Coding

The aim of this portion of the algorithm is to approximate the magnitude spectrum of each speech segment by a series of straight line approximations. By so doing, a compact description of the segment, which decreases storage requirements and facilitates feature extraction, is attained.

The algorithm initially used, but later discarded, was proposed by La Garde [1985]. Its features and implementation is described below.

5.9.1 The NEERI Approach

The NEERI Extrema coding algorithm [La Garde,1985] makes use of the fact that not all samples of a waveform or spectrum are of equal importance for its reproduction. If an algorithm can be found to select the most important samples, only these data points need to be coded.

The algorithm has been designed for use on both time domain waveforms and frequency domain spectra. In this application it has been optimized to code the magnitude spectrum of each speech segment.

The general coding scheme proposed involves the following procedure:

- beginning at the last coded extrenum, construct a straight line to the next candidate extrenum (this is actually two samples away as if the very next sample is selected as an

extrema no coding is achieved)

- calculate the distortion of the straight line approximation with respect to the portion of the original curve that is being approximated
- from the same start extrenum draw the next straight line to the sample a distance of 3 sample periods ahead of the start sample
- calculate the distortion of this line
- repeat this process until the straight line approximation has been made for each of 18 consecutive samples
- the sample that is chosen as the next extrema must satisfy the requirements that it be the sample with the furthest distance from the start sample while still having a distortion measurement better than a predetermined value
- its amplitude and distance from the previous extrema (TBE or Time Between Extrema) are then coded and stored
- the spectrum or waveform is then reconstructed by means of lineat interpolation between successive extrema

This coding schema was implemented in simulation and on the TMS 32020. Analysis of the performance indicated that the calculation of the distortion figure for each of the candidate extrenum proved to be very time consuming (in the order of 40% of real time for worst case approximations) for real time implementation. As a result an alternative method was required which would ideally satisfy both the constraints of real time implementation as well as good curve approximation.

5.9.2 The Wall Algorithm

Wall and Danielsson [1984] have proposed a sequential method of curve approximation based on area deviation. Since the method is sequential, the approximations are not optimal, but they are generated quickly.

The algorithm works by merging points one after another to the initial one, until a certain test criterion is no longer

satisfied. This criterion involves a parameter T, which can be set to different values. Greater values of T create longer segments on the penalty of a poorer approximation. The resulting output will be as in the NBERI algorithm an amplitude value of the last point which passes the test as well as the distance between the two successive extrema.

This method appears to be similar to the NBERI approach, however, they differ fundamentally in the test procedures used to select the extremum. The NBERI algorithm utilizes the basic SNR formula

$$\text{Distortion} = \frac{\Sigma(X-Y)^2}{\Sigma Y^2}$$

Thus for each trial point the following operations have to be carried out:

- a straight line has to be constructed between the last extrema and the trial point
- the difference between the actual curve points and the straight line has to be determined and each value squared and accumulated
- the original points on the curve have to be squared and accumulated
- the two summations above have to be divided to yield the distortion measure

This process is repeated for 18 trial points and if no failure occurs the eighteenth point is set to be the extrema.

The Pascal simulation procedure of the Wall algorithm is given as follows. Assume that a pointer PTR is used to point to the last extrema in a buffer (assume that this buffer is represented by the array BUF).

```
Procedure Calculate_Distortion;
var
  START   : integer;
  L,T     : real;
  FI      : integer;
  DELTA_FI: integer;
  FOUND   : boolean;
begin
  START := BUF [PTR];
  FI := 0;
  I := 1;
  FOUND := FALSE;
  repeat
    I := I+1;
    DELTA_FI:=I*(BUF[PTR+I]-BUF[PTR+I-1])-(BUF[PTR+I]-START)
    FI := FI + DELTA_FI;
    L := sqr(BUF[PTR+I]-START)+sqr(I);
    if sqr(T)*L < abs(FI) then
      begin
        FOUND := TRUE;
        I := I - 1;
      end;
  until FOUND;
end;
```

The value of I thus defines the distance between the extrema and the value BUF[PTR+I] is the amplitude value of the extrema.

A program was developed to determine the optimal values of T for voiced and unvoiced speech spectrums. The original spectrum and approximations with various values of T for both voiced and unvoiced segments is shown in figure 5.11. The optimal values determined by these procedures were

T = 1,5 for voiced segments

and T = 0,6 for unvoiced segments.

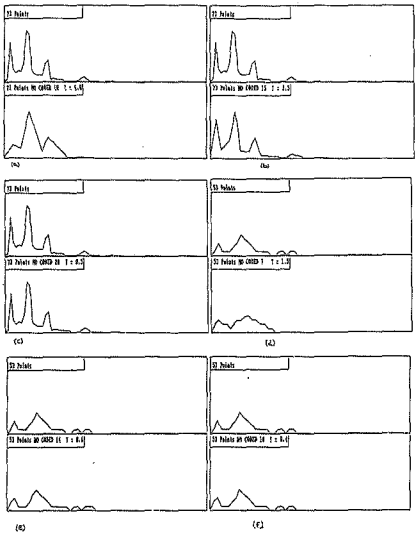

Figure 5.11 Extrema coding using the Wall algorithm for voiced segments with the following values of T (a) T = 5,0 (b) T = 1,5 (c) T = 0,5 and for unvoiced segments with (d) T = 1,5 (e) T = 0,6 (f) T = 0,4

5.9.3 Implementation of the Wall Algorithm

The algorithm is very simple and can be implemented efficiently on the TMS32020. The implemented algorithm description is as follows:

```
begin
  load the segment samples
  repeat
    repeat
      determine distortion of selected point
    until point passes or 18 points have been selected
    output the extrema and its distance value
  until end of segment
end.
```

A subroutine called LDSEG is used to load the segment data points into internal RAM. These points are located at address 200H. A variable called PTR is then set equal to this value.

A routine OUTPNT is provided to output the data addressed by PTR followed by the distance parameter which is found in the variable I.

The distortion calculation algorithm is implemented as follows:

```
begin
  set AR1 = PTR
  load data @AR1 and increment
  store this point in START ie. the origin
  set FI = 0
  set I = 1
  repeat
    increment I

    -- calculate DELTA_FI
```

```
multiply I with data @AR1 and increment
subtract from accumulator
multiply I with data @AR1
add to accumulator
subtract data @AR1

-- calculate FI

add FI
store accumulator in FI

-- calculate L

load data @AR1
subtract START
store in L
zero accumulator
square L and accumulate
square I and accumulate
store accumulator in L

-- determine T

if speech is voiced then
  T = 1.5 in Q4 format
else
  T = 0.6 in Q4 format
end if
```

```
        -- check if -(T*L)+FI > 0

        zero accumulator
        multiply T with L
        square FI and subtract P register from accumulator
        add P register to accumulator with a 4 bit left shift
        if accumulator > 0 then
          decrement I
        else
          check if end of search block is complete
        end if
      until search passes or end of block
    end.
```

The value of I thus specifies the spatial difference between the last extrema and the new extrema.

5.9.4 Analysis of Performance

This algorithm was implemented on the TMS 32020 and was found to execute in 4% of real time. This was a significant improvement over the NEERI algorithm implementation. The waveform approximations do not appear to suffer excessively from the simpler method and initial audible tests of digitized and coded waveforms suggest that the method is acceptable.

5.10 Summary

This chapter has described the implementation of a coding technique based on the method proposed by NEERI. The real time performance of the algorithm was tested and the following results were attained : the silence extraction portion of the algorithm was found to execute at worst case in 5,8% of real time; the segmentation routine consumed 15% of real time if a complete wide, broad and narrow search took place without valid pitch detection i.e. the worst case. For normal pitch detection 8% of real time was consumed and the Fast Pitch mode was executed in 4%

of real time; the Fourier Transform and segment comparison portions of the algorithms all functioned at worst case in 30% and 1,5% of real time, respectively. The Extrema Coding algorithm proposed by NEERI was discarded in favour of a simplified sequential algorithm which executed in 4% of real time.

CHAPTER 6 : CONCLUSION

The purpose of this chapter is to detail the achievements of this project as well to describe the future work for the next phase of development.

6.1 Overview of Progress

The project has been concluded at the stage where the bulk of the critical portions of the coding algorithm have been implemented on the TMS32020. The implementation of the short coding and run length coding portions have not been described in this project report due to time constraints, however, they are both based on simple algorithms which when implemented and included in the overall algorithm should not contribute significantly to the execution time.

6.2 Analysis of Execution Time and Speech Quality

The implementation of the algorithm has been described in the above report, however, such a project would be meaningless without attempting to analyse the quality, compression characteristics and real time performance of the algorithm.

The real time performance data has been presented for each portion of the algorithm. The sum of the worst case execution times adds up to a total of 56,3% of real time. This pessimistic statistic does, however, imply that the algorithm can definitely be implemented in real time. This figure will decrease significantly once the various portions of the algorithm are integrated into a single executable program as the time needed for data transfers will be diminished.

CHAPTER 6 : CONCLUSION

The purpose of this chapter is to detail the achievements of this project as well to describe the future work for the next phase of development.

6.1 Overview of Progress

The project has been concluded at the stage where the bulk of the critical portions of the coding algorithm have been implemented on the TMS32020. The implementation of the short coding and run length coding portions have not been described in this project report due to time constraints, however, they are both based on simple algorithms which when implemented and included in the overall algorithm should not contribute significantly to the execution time.

6.2 Analysis of Execution Time and Speech Quality

The implementation of the algorithm has been described in the above report, however, such a project would be meaningless without attempting to analyse the quality, compression characteristics and real time performance of the algorithm.

The real time performance data has been presented for each portion of the algorithm. The sum of the worst case execution times adds up to a total of 56,3% of real time. This pessimistic statistic does, however, imply that the algorithm can definitely be implemented in real time. This figure will decrease significantly once the various portions of the algorithm are integrated into a single executable program as the time needed for data transfers will be diminished.

The speech quality and compression characteristics were tested by coding and decoding a number of speech files. The original and coded/decoded versions of these files were then subjectively evaluated by the designer and a number of his colleagues. The overall conclusion was that although the speech was intelligible, speaker recognition was in a number of cases, not possible. This fact has been attributed to the problem of phase injection during the decoding phase. This conclusion was attained by comparing the speech quality of a sentence which was merely transformed into the frequency domain, the phase discarded and then inverse transformed, with a sentence that was coded with the entire process. The quality difference appeared to be very slight.

The compression statistics of the entire coding algorithm indicated that bit rates as low as 2 kbits/s were attainable, depending on the quality of the input speech. The average bit rate of the sentences tested was approximately 3 kbits/s.

6.3 Achievments of this Project

The main task of implementing an algorithm based on the proposal by NEERI has been achieved. In doing so it has been proved that it is possible to implement a complex speech coding algorithm on the TMS32020 digital signal processor.

During the course of the algorithm coding, numerous real time problems associated with the NEERI algorithm were encountered. It was thus the task of the designer to find alternative solutions which fell within the real time constraints. As a result research was conducted into developing the Transform and Extrema coding portions of the algorithm. In addition, many of the remaining portions had to be modified in various ways.

The end result being a number of separate programs each implementing a portion of the coding algorithm. The complete coding algorithm is tested by executing each of these portions in a sequential fashion.

6.4 Future Work

This project forms the cornerstone of the overall coding algorithm development. It has demonstrated that the implementation of a complex speech coding algorithm using state of the art technology, is now feasible.

In the next phase of development each portion of the algorithm will have to be analysed in conjunction with audible tests to determine their respective influences on the overall speech quality. The results of this analysis will then serve as the basis for a second design iteration.

The most difficult problem encountered during the course of this project was the subject of phase injection or alternatively phase preservation. This subject will have to be thoroughly investigated.

Finally, once the speech quality has been determined to be acceptable, the entire algorithm should be integrated to form a single executable program. The performance of the software should then be evaluated using a real time emulator.

REFERENCES

1. Adams, W.T and Brady J. (1983) Magnitude Approximations for Microprocessor Implementation, IEEE Micro, October 83, pp. 27-31

2. Burrus, C.S and Eschenbacher, P.W (1981) An In-phase, In-order Prime Factor FFT Algorithm, IEEE Trans. Acoust. Speech and Sig. Proc., ASSP-29, pp. 806-817

3. Crochiere, R.E; Cox, R.V and Johnston, J.D (1982) Real time Speech Coding, IEEE Trans. Comm., COM-30, No. 4

4. Gold, B. and Tierney J. (1983) Comments on Real Time Speech Coding, IEEE Trans. Comm., COM-31, No. 3, pp. 466-467

5. Good, I.J (1971) The relationship between the two fast Fourier transforms, IEEE Trans. Comput., C-20, pp. 310-317

6. Holmes, J.N (1982) A survey of methods for digitally encoding speech signals, The Radio and Electronic Engineer, Vol 52, No. 6, pp. 267-276

7. Jayant, N.S (1974) Digital Coding of Speech Waveforms : PCM, DPCM, DM quantizers, Proc. IEEE, Vol 62, pp. 611-632

8. La Garde, P. (1985) Speech Processing using Extrema Coding, paper presented at the South African symposium on digital signal processing, Stellenbosch

9. Maitra, S. and Davis, C.R (1979) A Speech Digitizer at 2400 Bits/s, IEEE Trans. Acoust. Speech and Sig. Proc., ASSP-27, no. 6, pp. 729-733

10. Miyamoto, T.; Inada, H. and Nakata, K. (1983) A Real time PARCOR Analysis of Speech by High Performance Signal Processors, Electronics and Communication in Japan, Vol. 56-A, no. 7

11. O'Neal Jr., J.B (1983) The Six Principles of Low Bit Rate Encoding, IEEE International Conference on Communications, Vol. 2, pp. 1142-1146

12. Quarmby, D.J and Holmes, J.N (1984) Implementation of a Parallel-formant Speech Synthesiser using a Single Chip Programmable Signal Processor, IEE Proceedings, Vol 131, Pt. F, no. 6

13. Rabiner, L.R; Cheng, M.J; Rosenberg, A.E and McGonegal C.A (1976) A comparitive performance study of several pitch detection algorithms, IEEE Trans. Acoust. Speech and Sig. Proc., ASS pp. 399-418

14. Rabiner, L.R and Schafer R.W (1978), Digital Processing of Speech Signals, Prentice Hall

15. Reeves, A.H (1938) French Patent 852183

16. Shannon, C.E (1948) A mathematical theory of communication, Bell Syst. Tech. Journal, 27, pp. 379-423 and 623-656

17. Texas Instruments (1985) TMS 32020 Users Guide

18. Thompson, J.S and Tewksbury, S.K (1982) LSI Signal Processors Architecture for Telecommunication Applications, IEEE Trans. Acoust Speech ans Sig. Proc., ASSP-30, no. 4

19. van Schalkwyk, J. (1985) Course notes on 'Digital coding of speech signals', presented at University of Pretoria

20. Wall, K. and Danielsson, P. (1984) A fast sequential method for polygonal approximation of digitized curves, Computer Vision, Graphics and Image Processing, 28, pp. 220-227

21. Webster, J.C (1981) Information in simple multidimensional speech messages, *Journal of the Acoustic Society of America*, 33, pp. 940-944